UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dinamicidad en Sistemas Multiagente basados en THOMAS

Autor: Alfonso Antonio Machado Benito
Director: Vicente J. Julián Inglada
Titulación: Ingeniería Informática
06, 2011

Agradecimientos

Agradezco a toda la gente que ha estado conmigo en estos últimos seis años. A todos mis amigos por todos esos buenos y malos momentos que hemos vivido. A Vicente Botti por ofrecerme la oportunidad de entrar a formar parte del GTI. La paciencia de Natalia, Elena, Juan Ángel, Sergio Pajares, Mario y especialmente Joan, por aguantar todas mis preguntas. A mis compañeros del *frikilab* por aguantarme como su compañero de laboratorio. A mi director, Vicente Julián, por toda la dedicación que ha tenido conmigo. A mi madre, que ha hecho que tenga que preocuparme durante toda la carrera de solamente estudiar y trabajar. Finalmente, a Leti, que ha estado conmigo en los peores y mejores momentos a lo largo de esos más de dos años y medio. Gracias.

Alfonso Machado

Índice general

Capítulo 1

Introducción y objetivos

Actualmente, hay una clara tendencia en el uso de métodos y herramientas que permiten desarrollar Sistemas Multiagente (MAS) capaces de ofrecer dinamicidad en las organizaciones, permitiéndoles así modificar su comportamiento al detectar cambios en el entorno. Las Organizaciones Virtuales (VO) [6] son entendidas como una forma de modelar sistemas tomando un punto de vista sociológico. Dichas organizaciones están basadas en el principio de cooperación en negocios sobre redes abiertas, el cual versa sobre la ayuda entre los participantes para obtener un objetivo común o mayor [3] [6]. La principal ventaja que obtenemos al aplicar este principio es que nos provee de elementos que nos aportan flexibilidad y, además, capacidad de respuesta rápida. Por todo ello obtenemos un mejor servicio y logramos en consecuencia una mayor satisfacción del cliente al obtener un mejor servicio.

Una cualidad a resaltar de las Organizaciones Virtuales cuando se usan Sistemas Multiagente (MAS) es la de que los sistemas son modelados con un nivel alto de abstracción. De este modo, la separación entre el mundo real y los modelos de sistemas se reduce considerablemente. Además, este tipo de sistemas facilita la implementación de sistemas abiertos y heterogéneos [11]. Las Organizaciones Virtuales no deben solamente permitir describir aspectos de la estructura del sistema (funciones, grupos de agentes, interacción y patrones de relación entre roles) y aspectos del comportamiento funcional (tareas de los agentes, planes o servicios), sino también describir el comportamiento de las normas que deben seguir los agentes, la entrada y salida de los componentes dinámicos y su formación, también dinámica. Dadas las características de estos sistemas abiertos, y atendiendo especialmente a su faceta de dinamismo, es esencial encontrar una aproximación que soporte la evolución de estos sistemas y facilite el crecimiento y actualización de ellos en tiempo de ejecución. De acuerdo a estas necesidades se ha desarrollado **THOMAS** (*MeTHods, Techniques and Tools for Open Multi-Agent Systems*) [5] [8] [1], una arquitectura MAS para Organizaciones Virtuales, especialmente enfocada a soportar dinamicidad en las organizaciones de agentes basada en la plataforma Magentix2 como *Platform Kernel* de la arquitectura THOMAS para soportar los agentes que se ejecutarán.

El objetivo de este **Proyecto Final de Carrera**, el cual se encuentra dentro del trabajo de investigación del proyecto Consolider Ingenio (CSD2007-00022), *Agreement Technologies* y del proyecto del Ministerio de Ciencia e Innovación (TIN2009-13839-C03-01) *Organizaciones Virtuales Adaptativas: Arquitecturas y Métodos de Desarrollo*, consistirá en demostrar que el binomio THOMAS/Magentix2 es una plataforma que permite ofrecer dinamicidad en los Sistemas Multiagente (MAS), constituyendo un paso, y la base, con la que se puede empezar a trabajar en Sistemas Adaptativos. Esto es, sistemas que por si solo son capaces de adaptarse a cualquier evento que ocu-

rra de tipo estructural (una organización desaparece, se deja de ofrecer un servicio, etc) y funcional (como, por ejemplo, nuevos servicios). Este objetivo principal se divide en los siguientes subojetivos:

- Estudio de la plataforma de Magentix2 como software que soporta Sistemas Multiagente.
- Estudio del framework THOMAS como software que ofrece Organizaciones Virtuales a los Sistemas Multiagente.
- Características que debe cumplir el framework para ofrecer dinamicidad en las Organizaciones Virtuales de los Sistemas Multiagente.
- Diseño de un caso de estudio junto con un conjunto de escenarios que verifiquen los diferentes aspectos que permiten la dinamicidad en las Organizaciones Virtuales.
- Implementación de los escenarios diseñados para verificar las propiedades dinámicas de las Organizaciones Virtuales.
- Ejecución de los diferentes escenarios diseñados para comprobar la dinamicidad de la plataforma comentada anteriormente.
- Ejecución y validación de los resultados obtenidos.

El caso de estudio que hemos desarrollado consiste en un mercado de turismo. Este mercado ofrece un entorno para que distintos agentes puedan crear agencias de viajes y ofrecer sus servicios (búsqueda de hoteles, vuelos, tren, etc), que cualquier agente pueda ofrecer determinado servicio y que los clientes puedan buscar y utilizar los distintos servicios ofrecidos por las agencias. Además, al usar la plataforma THOMAS/Magentix2 se pretende verificar que todos los métodos funcionan correctamente, que la plataforma es estable, que la comunicación entre ella y otras plataformas funciona, encontrar posibles errores, mejorar aquellos aspectos que se consideren oportunos desde el punto de vista del desarrollador y también verificar la especificación de THOMAS/Magentix2.

En primer lugar, en el capitulo 2 haremos una breve explicación de que son los Sistemas Multiagente, Organizaciones Virtuales y explicaremos las tecnologías THOMAS y Magentix2. En la sección 3.1, explicaremos lo que es dinamicidad, cómo conseguirla y qué esperamos que tengan las tecnologías anteriores para que se pueda decir que ofrecen dinamicidad en los Sistemas Multiagente. A continuación, en la sección 3.2 haremos una descripción del caso de estudio, Mercado de Turismo, donde explicaremos los agentes que

intervienen y la estructura que hemos utilizado. Seguidamente, en la sección 3.3 describiremos los distintos escenarios y explicaremos las pruebas que se han realizado. Por último comentaremos y valoraremos si hemos alcanzado o no lo objetivos propuestos y qué conclusiones hemos obtenido.

Capítulo 2

Plataforma para Sistemas Multiagente

2.1. Sistemas Multiagente

Los **Sistemas Multiagente** (MAS) [12] son sistemas compuestos por múltiples agentes inteligentes que interactuan entre sí. Son una evolución de los sistemas distribuidos inteligentes y se usan para resolver problemas que son muy difíciles o imposibles de resolver por sistemas monoproceso. Se suelen usar para modelar sistemas de comercio on-line, simulación de desastres naturales, comportamientos sociales, de manadas de animales, cadenas de montaje, videojuegos, etc.

El elemento principal de un Sistema Multiagente (MAS) son los agentes. Aunque no existe una definición clara, un **agente** [12] es una entidad inteligente que existe dentro de un determinado ambiente y que se comunica a través de un sistema de comunicación usando un protocolo de comunicación. Podría verse como que los agentes son los nodos inteligentes de un sistema distribuido donde el trabajo en conjunto produce un resultado.

Las plataformas de Sistemas Multiagente (MAS) son la base para construir sistemas mas complejos. Nuestro objetivo es ir más allá: es ayudarnos de una plataforma de Sistemas Multiagente para contruir sistemas adaptativos y para ello tenemos que verificar si la plataforma que usamos ofrece dinamicdad.

Actualmente la plataforma más usada en investigación es *Java Agent DEvelopment Framework*[1] (**JADE**), que implemente el estándar **FIPA**[2] y el protocolo de comunicación **FIPA-ACL**[3].

2.2. Organizaciones Virtuales

Organizaciones Virtuales (VO) son un conjunto de individuos e instituciones que necesitan coordinar recursos y servicios a través de las fronteras institucionales [7] [4]. Son sistemas abiertos formados por la agrupación y colaboración de entidades heterogéneas. Además, existe una separación entre la forma y la función que requiere la definición de cómo se llevará acabo un determinado comportamiento. La tecnología de los Sistemas Multiagente (MAS) y las organizaciones dinámicas de agentes son especialmente adecuadas como soporte de estos sistemas abiertos. Las principales características de las Organizaciones Virtuales son:

[1] Más información en: http://jade.tilab.com/
[2] Véase: http://www.fipa.org/
[3] Véase: http://www.fipa.org/repository/aclspecs.html

- Están formadas por agentes heterogéneos que pueden salir y entrar del sistema dinámicamente.

- Se sitúan en entornos dinámicos.

Como consecuencia de la heterogeneidad de estos sistemas, son necesarios conceptos organizacionales para coordinarlos. Las funcionalidades del sistema deben ser modeladas como servicios con el fin de permitir a la heterogeneidad de agentes u otras entidades que interactuen con el sistema de una forma estandarizada. Hay que tener en cuenta los posibles cambios en los entornos dinámicos podrían requerir la adaptación de las estructuras y funciones de las organizaciones. Por otra parte, esta necesidad de adaptación del sistema a desarrollar requiere la aplicación de conocimientos de nuevas técnicas de razonamientos y mecanismos de planificación para resolverlo de una forma dinámica.

La integración de Sistemas Multiagente (MAS) y los Servicios Web [9] ha sido propuesta como la base de estos nuevos y complejos sistemas. Esto permite que ambas tecnologías puedan complementarse y aportar los puntos fuertes de cada una:

- Los estándares en los servicios proporcionan una infraestructura para la interacción entre los agentes.

- Los Sistemas Multiangte (MAS) ofrecen una noción mas general y compleja de la Arquitectura Orientada a Servicios (SOA).

- La capacidad social e inteligente de los agentes permite definir servicios mas complejos.

Sin embargo, de acuerdo con las investigaciones actuales, es necesario mejorar los métodos de coordinación en las Organizaciones Virtuales por medio de técnicas de localización y composición de servicios, tanto sintácticamente como semánticamente. La existencia de un mecanismo de adaptación para la creación de la estructura de las organizaciones permite optimizar la coordinación de las mismas, teniendo en cuenta la heterogeneidad de los agentes y los servicios. Así surge el requisito de emplear mecanismos básicos que permitan encontrar servicios adecuados en Organizaciones Virtualels (OV) y, si se necesitase, construir nuevos servicios usando composición de los mismos. También la existencia de mecanismos de regulación que garanticen una coordinación global de sistemas abiertos eficiente, pero teniendo en cuenta la imposibilidad de controlar directamente (en su gran mayoría) los agentes y servicios.

En la sección 2.3 y 2.4 de esta memoria explicaremos las arquitecturas de THOMAS y de Magentix2 que están específicamente diseñados para la ejecución de Organizaciones Virtuales (VO).

2.3. THOMAS

MeTHods, Techniques and Tools for Open MultiAgent Systems (THOMAS) es una arquitectura [5] abierta que permite crear los elementos básicos para agrupar agentes inteligentes en Organizaciones Virtuales (OV).

La información y mecanismos ofrecidos por el tradicional FIPA *Directory Facilitator* (DF) no es suficiente para hacer frente a los sistemas abiertos y dinámicos. Por tanto, es necesario desarrollar métodos inteligentes para hacer frente a la gestión de servicios en Sistemas Multiagente (MAS) abiertos y desentralizados. Por ejemplo servicios de localización inteligente teniendo en cuenta la información semantica o generación y adaptación de composición de servicios. En una arquitectura FIPA[4] típica, el DF tradicional tiene las siguientes limitaciones:

- La descripción de servicios es muy básica (nombre, tipo, protocolo, ontología, lenguaje, propietario, propiedades, etc).
- No considera organizaciones.
- Está solamente orientado al paradigma de agentes.
- Sólo permite funcionalidades básicas (registrar, borrar, modificar y buscar)
- El algoritmo de descubrimiento de servicios es muy simple ya que el mecanismo por defecto es busqueda en profundidad (DFS) y la información semántica no se tiene en cuenta.
- El descubrimiento de servicios se limita a descubrir servicios simples ya que el algoritmo por defecto no considera la composición de servicios.

Por tanto, la arquitectura THOMAS se alimenta de la arquitectura FIPA pero extiende sus principales componentes (*Agent Managemnt System* (AMS) y *Directory Facilitator* (DF)) en *Organization Management System* (OMS) y *Service Facilitator* (SF) respectivamente. Los principales componentes de la arquitectura THOMAS son *Platrform Kernel* (PF), *Organization Management System* (OMS) y *Service Facilitator* (SF)

[4] Véase: http://www.fipa.org/

2.3.1. Platform Kernel

Se ocupa de los servicios básicos de la administración de los agentes y lo puede proporcionar cualquier plataforma compatible con FIPA. La función de este componente es ofrecer el ciclo de vida de los agentes y la comunicación entre ellos. En nuestro caso la plataforma usada es Magentix2 la cual explicaremos en la sección 2.4.

2.3.2. Organization Management System

Se encarga de la gestión de las Organizaciones Virtuales (OV) y tiene en cuenta el control de su estructura subyacente, los roles que desempeñan los agentes dentro de la organización y las normas que rigen el comportamiento del sistema.

El *Organization Management System* (OMS) es el encargado de controlar la creación de los grupos de agentes, llamados *Organizational Units* [2], qué entidades participan dentro de la unidad, cómo estas entidades se relacionan entre cada una y qué roles desempeña cada entidad en cada momento. Por lo tanto, el OMS provee a los agentes un conjunto de servicios para la gestión del ciclo de vida de las organizaciones, clasificados en:

Servicios estructurales

Modifican la estructura y la normativas de las especificación de una organización(roles, normas y unidades).

- **Registra unidad**. Un agente ejecuta este método para registrar una unidad en el OMS. Este método añade una nueva unidad a la lista de unidades activas a las cuales pueden entrar y salir nuevos agentes. Además permite registrar nuevas unidades que hereden de esta. Si un agente quiere invocar este funcionalidad debe invocar el método *RegisterUnit* al cual se le pasa como parámetro el nombre de la unidad a crear, el tipo de unidad, descripción de la unidad y de qué unidad hereda. La forma de invocarlo sería de la siguiente manera:

```
String result = null;
try {
    // RegisterUnit Principal: TourismMarket
    result = OMSservices.registerUnit("TourismMarket",
                "hierarchy",
                "Ofrecer Servicios de Turismo",
                "virtual");
} catch (Exception e) {
    logger.error(e.getMessage());}
```

- **Borrar unidad**. Elimina una determinada unidad de la lista de unidades activas, sólo si el agente que la crea tiene el rol `creator` y es el único agente dentro de la unidad. Borrar una unidad del sistema implica que ha habido un cambio en la estructura del sistema como que una entidad ha sido borrada debido a algún evento. Esto es un claro ejemplo de las posibilidades un sistema dinámico. Para borrar una unidad hay que usar el método *DeregisterUnit* al cual se le pasa por parámetro el identificador de la unidad. La ejecución de este método se haría de la siguiente forma:

```
String result = null;
try {
    // Deregister main unit: TourismMarket
    result = OMSservices.DeregisterUnit("TourismMarket");
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Registrar rol**. Registrar un rol dentro de una organización añade un nuevo rol a la lista de roles que tiene una organización asociados. Esto permite que los agentes puedan adquirir un papel dentro de la organización el cual tendrá asociado una serie de características como se ha mencionado anteriormente. Para registrar un rol hay que ejecutar el método *registerRol* pasandole por parámetro el nombre del rol que se desea crear, la unidad a la que pertenecerá, el tipo de visibilidad, la posición del rol en la unidad, la privacidad del rol y de que rol hereda. La forma de ejecutarlo sería como a continuación:

```
String result = null;
try {
    // Register new role: MarketManager
    result = OMSservices.registerRole("MarketManager",
      "TourismMarket","external","supervisor","private",
      "member");
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Borrar rol**. Elimina el correspondiente rol de la lista de roles que tiene asociado una organización. A partir de ese momento ningún otro agente podrá desempeñar ese rol. Para poder realizar esta acción, no tiene que haber ningún agente que desempeñe ese rol, ni ninguna norma que haga referencia a ese rol, ya que dejaría el sistema inconsistente. Para borrar un rol se tiene que ejecutar el método *deregisterRole* al que se le pasa

por parámetro el identificador del rol que se desea eliminar. Se ejecuta como a continuación:

```
String result = null;
try {
    // Deregister role: MarketManager
    result = OMSservices.DeregisterRole("MarketManager");
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Registrar norma**. Añadir un nueva norma al sistema. El método a usar es *registerNorm* y añade una norma a la lista activa de normas del sistema. Cuando la norma se añade tiene que cumplirse a partir de ese momento. Al método se le pasa un identificador de la norma a añadir y la norma a añadir. La forma de invocar el método es la siguiente:

```
String result = null;
try {
    // Register a norm: Only one Payee
    result = OMSservices.registerNorm("norm1"
  "FORBBIDEN...acquireRole_MESSAGE(...(ROLE_'Payee'))");
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Borrar norma**. Eleminar una norma de la lista de normas activas del sistema. El método es *deregisterNorm* y se le pasa como parámetro el identificador de la norma a eliminar. La forma de usarlo sería:

```
String result = null;
try {
    // Deregister norm: only one Payee
    result = OMSservices.deregisterNorm("norm1");
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

Servicios informativos

Son servicios que dan información sobre el estado de las organizaciones.

- **Informe de agente**. Este método permite saber que roles y en que unidad desempeña un agente en ese momento. El método es *informAgentRole* y se le pasa por parámetro el identificar del agente que se desea saber esa información. Un ejemplo de la ejecución sería:

```
ArrayList<String> result = null;
try{
    result=informAgentRole(godAgent.getAid().toString());
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Informe de miembros**. Un agente ejecuta este método cuando desea conocer las entidades que pertenecen a una unidad. Se le puede indicar un rol de la unidad y solo las entidades que desempeñen ese rol se le mostrará su información. Un agente solo podrá usar este servicio dependiendo del tipo de unidad: si es `FLAT` el agente podrá usarlo, si es `TEAM` solo podrá hacerlo si pertenece a la organización y si es `HIERARCHY` sólo podrá si desempeña el rol `supervisor`. El método es *informMembers* y se le pasa el rol y la unidad del que se desean conocer sus entidades. Un ejemplo de su ejecución es:

```
ArrayList<String> result = null;
try{
    result = informMembers("MarketMananger",
                           "TourismMarket");
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Informe de normas**. Un agente invoca este método cuando desea conocer las normas que están asociadas a un determinado rol. El método que tiene que ejecutar el agente es *informRolesNorms* y recibe por parámetro el identificador del rol que quiere conocer las normas asociadas a él. Una forma de usarlo sería como a continuación:

```
ArrayList<String> result = null;
try{
    result = informRolesNorms("Payee");
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Informe sobre *profile***. Este método permite conocer la descripción de los servicios que están asociados a un determinado rol. El agente que quiera usarlo tiene que invocar el método *informRoleProfile* y le pasa como parámetro el identificador del rol del cual desea conocer la descripción de los servicios asociados a ese rol.

```
ArrayList<String> result = null;
try{
    result = informRoleProfiles("MarketManager");
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Informe de unidad**. Un agente ejecuta este método cuando desea conocer la descripción de una unidad. El método es *informUnit* y se le pasa por parámetro la unidad de la que se desea conocer sus detalles. Una posible implementación sería:

```
ArrayList<String> result = null;
try{
    result = informUnit("Agency1");
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Informe de unidades y roles**. Este método es usado por un agente cuando desea conocer la lista de roles que han sido registrados dentro de una unidad. Un agente podrá invocar este método dependiendo del tipo de unidad: si la unidad es `FLAT` podrá hacerlo y si es `TEAM` o `HIERARCHY` si y sólo si es miembro de la unidad. El método es *informUnitRoles* y se le pasa por parámetro la unidad de la que se desea conocer la lista de roles que están asociados a ella. Una posible invocación del método sería:

```
ArrayList<String> result = null;
try{
    result = informUnitRoles("MarketManager")
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

Servicios dinámicos

Estos servicios permiten la gestión de la entrada y salida de los agentes en el sistema.

- **Adquirir rol**. Desempeñar un rol dentro de una organización implica que el agente que va a realizar esta acción podrá realizar ciertas tareas asociadas a ese rol, y/o que va a ocupar cierto rango dentro de la

organización. Un agente puede tener varios roles, pudiendo desempeñar varias labores dentro de una organización o incluso poder tener distintos rangos en el sistema. El método que lo hace es *acquireRol* y se le pasa por parámetro el rol que quiere desempeñar y la unidad en la cual va a adquirir ese rol. La forma de usarlo es como a continuación:

```
String result = null;
try {
    //Acquire MarketManager role inside TourismMarket unit
    result = OMSservices.acquireRole("MarketManager",
                                     "TourismMarket");
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Dejar rol**. Dejar de desempeñar un papel dentro de una organización implica que un agente ha decidido dejar de jugar un rol dentro de una unidad, y todo lo que con ello implique, como cumplir ciertas normas, o poder ofrecer determinados servicios. El método que permite esto es *leaveRole* y se le pasa por parámetro el rol que se quiere dejar de jugar y la unidad en la que está ese rol. La forma de usarlo es como a continuación:

```
String result = null;
try {
    // Dejar
    result = OMSservices.leaveRole("member", "virtual");
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

2.3.3. Service Facilitator

Es el encargado de gestionar los servicios para facilitan su publicación y descubrimiento de éstos por los clientes potenciales. Para una mayor comprensión, puede ser entendido como un servidor de páginas amarillas.

El SF es una redefinición del DF tradicional de FIPA, siendo ahora capaz de hacer frente a los servicios de un modo mas trabajado, siguiendo las directrices de la Arquitectura Orientada a Servicios (SOA). Hace frente a la limitación del tradicional Directory Facilitator (DF) de FIPA considerando información semántica, composición de servicios, así como los roles, objetivos y duración de los servicios.

El SF proporciona un conjunto de servicios estándares para la gestión de las funcionalidades de organizaciones y agentes. El SF está clasificado en dos partes: registro y descubrimiento.

Registro

Estos servicios permiten añadir, modificar y eliminar servicios del directorio del *Service Facilitator* (SF).

- **Registar *profile***. El añadir una descripción de servicio (*Profile Description*) en el *Service Facilitator* (SF) conlleva, añadir una nueva entrada en el SF de la lista de servicios que se necesitan, usan o requiere el sistema. Añadir la descripción del servicio implica que se pueda buscar el servicio y averiguar si alguien lo ofrece o no, para ejecutarlo o añadirse como proveedor. Para añadir un *profile* al SF se ejecuta el método *registerProfile* que se le pasa como parámetro el *profile* del servicio. El método se ejecuta de la siguiente forma:

```
String result = null;
try {
    result = SFservices.registerProfile(NewClientProfile);
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Eliminar *profile***. Este método se usará cuando un agente el cual había necesitado la implementación de un servicio que él no podía ofrecer, decida borrarlo porque ya no lo necesita, o porque alguien en el sistema considere que ya no es necesario. Ejecutar este método elimina un *Profile Description* de la lista de servicios ofrecidos en el sistema. Al eliminarse no se podrá encontrar ninguna posible descripción de ese servicio. Solo se podrá eliminar el servicio si no hay ningún proveedor. El método a usar sería *deregisterProfile* y se le pasa como parámetro el *profile* que se desea borrar. Un posible ejemplo de su uso sería:

```
String result = null;
try {
    result=SFservices.deregisterProfile(NewClientProfile);
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Añadir *process***. Añade una implementación del servicio, descrito en el *profile* asociado, que ofrece el agente que ejecuta el servicio. La eje-

cución de este método implica que cuando se quiera saber quien implementa determinado servicio, aparezca el agente que ha registrado la implementación del servicio (*Process Description*). El método a usar es *registerProcess* y se le pasa como argumentos el *process* asociado al profile que se quiere añadir. Un posible uso de este método sería:

```
NewClientProcess.setProfileID(serviceId);
try {
    result=SFservices.registerProcess(NewClientProcess);
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Eleminar *process***. Este método elimina una implementación (*Process Description*) de un servicio que esté registrado en el *Service Facilitator* (SF), que era ofrecida por un agente. Al ejecutarse el método, se elimina al agente que lo ejecuta y a la implementación del servicio de la lista de proveedores del *profile* asociados al *process*. El método que se tiene que usar es *removeProvider* y se le pasa como parámetro el *process* que se quiere eliminar. La forma de usarlo sería como se muestra a continuación:

```
try {
    result = SFservices.removeProvider(NewClientProcess);
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

Descubrimiento

Este conjunto de servicios buscan y componen servicios como respuesta de las necesidades del usuario.

- **Buscar servicio**. Este método es ejecutado por un agente cuando desea encontrar algún servicio en el sistema. El sistema de búsqueda que se usa es una búsqueda sintáctica, es decir, aquellos servicios que tengan relación en su descripción de servicio con lo que uno desee buscar serán los que se le devuelvan al agente que ha ejecutado la búsqueda. El método devuelve al agente una lista de *Profile Description* donde cada uno de los elementos de esa lista corresponde con un tipo de servicio y una descripción del mismo. Un ejemplo de uso se muestra a continuación:

```
ArrayList<String> outputs = new ArrayList<String>();
String serviceId = null;
String result = null;
try {
    outputs = SFservices.searchService("NewClient");
} catch (Exception e) {
    logger.error(e.getMessage());
}
if (outputs.size() == 0) {
    // Not find the service
} else {
    // The service is found
    // The service id of the first  element
    serviceId = outputs.get(0);
}
```

- **Obtener *profile***. Un agente ejecuta este método cuando desea conocer el *profile* de un servicio determinado. El método a ejecutar es *getProfile* y se le pasa por parámetro el identificador del servicio que se quiere conocer su *profile*. La forma de usarlo sería como a continuación:

```
try {
    URLProfile = SFservices.getProfile(45);
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

- **Obtener *process***. Este método se ejecuta cuando un agente desea conocer el *process* de un servicio determinado para seleccionar un proveedor al que solicitar el servicio. El método a ejecutar es *getProcess* y se le pasa por parámetro el identificador del *profile* del servicio que se quiere conocer su *profile*. La forma de usarlo este método sería como a continuación:

```
try {
    providers = SFservices.getProcess(34);
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

2.4. Magentix2

Magentix2 es una plataforma de agentes para Sistemas Multiagente abiertos. Su principal objetivo es acercar esta tecnología a dominios reales: nego-

cio, industria, logística, e-comercio, salud, etc. La evolución tecnológica en el área de la Tecnología de la Informática y la Comunicación ha planteado nuevos retos y requisitos en los sistemas software. Además, la informática se concibe como una parte inherente en la sociedad actual. La tecnología de los Sistemas Multiagente tiene algunas características que indican su potencial para soportar este nuevo paradigma. La dinamicidad en las organizaciones de los agentes, los cuales son capaces de adaptar automáticamente su comportamiento con el fin de aprovechar al máximo su entorno en un momento dado, se está convirtiendo cada vez en algo mas importante. Los factores sociales en las organizaciones de los Sistemas Multiagente son cada vez más importantes en la interacción de la estructura entre los mundos dinámicos y abiertos

El proyecto Magentix2 es una continuación del proyecto Magentix, cuyo objetivo final es extender las funcionalidades de Magentix, ofreciendo nuevos servicios y herramientas que permitan seguridad y optimizar la gestión de Sistemas Multiagente abiertos.

Por tanto, el objetivo principal Magentix2 es desarrollar una tecnología que haga frente a la alta dinamicidad en la topología de un sistema y con interacciones flexibles, las cuales son consecuencia natural de los sistemas distribuidos y autónomos de sus componentes. En este sentido, la plataforma se ha ampliado con el fin de soportar la interacción de los protocolos y las conversaciones, y las interacciones entre las organizaciones de agentes. Finalmente, otro aspecto importante cubiertos por Magentix2 es la seguridad.

Así hemos desarrollado un modelo de seguridad que incorpora mecanismos de seguridad a bajo nivel y medidas de confianza que complementan los métodos de criptografía clásica. Entre otras funcionalidades, Magentix2 incorpora un servicio de traza. Este servicio permite a los agentes publicar o suscribirse a eventos (cambio en algún atributo, recepción de un mensajes, etc). Por último, Magentix2 incorpora el framework THOMAS, permitiendo a los usuarios gestionar aspectos de organización y servicio con facilidad.

Magetinx2 ofrece soporte a tres niveles: organizaciones, interacciones y agentes.

- **Organización**. Tecnología y técnicas relacionadas con las sociedades de agentes.
- **Interacción**. Tecnología y técnicas relacionadas con la comunicación entre agentes.
- **Agente**. Tecnología y técnicas relacionadas con el propio agente (como razonamiento y aprendizaje).

Figura 2.1: Componentes de Magentix2.

Con el fin de ofrecer estos niveles de soporte, Magentix2 está formado por diferentes componentes (como se puede ver en la imagen 2.1), y ofrece tecnologías para el desarrollo y ejecución de Sistemas Multiagente.

2.4.1. Infraestructura de comunicación

Magentix2 usa *AMQP*[5] como protocolo de comunicación entre agentes. Este estándar abierto está diseñado para soportar mensajería segura y de alto rendimiento a través de Internet. Facilita la interoperabilidad entre entidades heterogéneas. Magentix2 permite que agentes heterogéneos interactúen entre sí a través de mensajes *FIPA-ACL*, que se intercambian a través del estándar de *AMQP*.

Concretamente, Magentix2 usa *Apache Qpid*[6] que implementa *AMQP*. Por tanto, los agentes Magentix2 usan la *APR* de clientes *Qpid* para conectarse al servidor *Qpid* y comunicarse con otros agentes a través de Internet.

[5]Más información en http://www.amqp.org/confluence/display/AMQP/Advanced+Message+Queuing+Protocol

[6]Más información en http://qpid.apache.org/

2.4.2. Servicio de traza

El servicio de traza permite a los agentes en un Sistema Multiagente compartir información de forma indirecta mediante traza de eventos. Este servicio está basado en el patrón de diseño de publicador/subscriptor, el cual permite subscribirse a un filtro de eventos atendiendo a algunos atributos (content-based filtering), ya que los agentes solo reciben la información que a ellos les interesa y sólo la información solicitada se transmite por la red.

Para facilitar la labor a la plataforma de agentes, Magentix2 incorpora un gestor de trazas (*Trace Manager*, TM), el cual se encarga de coordinar el proceso de traza de eventos, permitiendo publicar o eliminar lo publicado, subscribirse o borrar la subscripción, trazar información, o buscar información de las trazas disponibles en tiempo de ejecución.

2.4.3. Seguridad

Magentix2 incorpora un modulo de seguridad el cual ofrece características como seguridad, privacidad, *openness* e interoperabilidad, las cuales no son ofrecidas por ninguna otra plataforma de agentes actualmente. Este módulo está basado en los estandares abiertos (AMPQ, SSL, SASL, certificados X.509, WS-Security, FIPA-ACL) y tecnologías de código abierto (Qpid, NSS, Axis2, Rampart). Este modulo de seguridad permite a los desarrolladores no sólo seguridad sino también una Sistema Multigante abierto el cual permite entrar y salir en cualquier momento a agentes que previamente eran desconocidos.

El principal componente del modulo de seguridad es el Servicio de gestión de Magentix2 (MMS), el cual es un *WS-Secutiry compliant secure web service* implementado usando Apache Axis2[7] y el modulo de seguridad Apache Rampart. El MMS es el encargado de controlar la creación de certificados de agentes para los usuarios de la plataforma. Estos usuarios deben identificarse a través del MMS con un certificado emitido por una autoridad tercera. Después, el MMS emite (usando la Autoridad Certificadora de Magentix2 (MCA)) un certificado para ese agente que le permite autenticarse dentro de la plataforma. Por tanto, los agentes pueden comunicarse entre ellos de un modo seguro cuando obtienen un certificado válido del MCA. Para ello, el *broker* Qpid está configurado para que acepte los certificados emitidos por el MCA.

[7] Véase: http://axis.apache.org/axis2/java/core/

2.4.4. Conversación entre agentes

Magentix2 incorpora agentes conversacionales, los cuales son una clase de agente llamada *CAgents*. Esta clase de agentes permite la creación automática de conversaciones simultáneas basada en protocolos de interacción. CAgents pueden usarse para predefinir protocolos de interacción, definiendo su propio protocolo de interacción y dinámicamente cambiar protocolos de interacción en tiempo de ejecución.

CAgents está compuesto por dos principales componentes: Factorías de Conversación (*CFactories*) y Procesadores de Conversación (*CProcessors*). *CFactory* define un protocolo de interacción como una máquina de estado finito mediante nodos y arcos entre ellos. *CFactories* gestiona automáticamente la entrada de mensajes, decidiendo si un mensaje pertenece a la conversación en curso o tiene que crearse una nueva. Sin embargo, *CFactories* permite a los agentes mantener varias conversaciones siguiendo simultáneamente el mismo protocolo de interacción y gestionar distintos aspectos a la vez.

2.4.5. Framework THOMAS

La plataforma Magentix2 no solo tiene como objetivo ofrecer un mecanismo al programador que garantice la comunicación entre agentes, sino también ofrecer un soporte completo para Organizaciones Virtuales[8] (OV). Con este propósito ha sido integrado el framework de THOMAS [13].

Los agentes tienes acceso a la infraestructura ofrecida por THOMAS a través de un conjunto de servicios incluidos en los dos principales componentes de la estructura de THOMAS: *Service Facilitator* (SF) y *Organization Management Service* (OMS).

Magentix2 implementa dos agentes intermediarios, SF y OMS, que ofrecen la API de THOMAS a los agentes de la plataforma de un modo sencillo. De esta forma, se ha conseguido un intermediario entre los agentes de Magentix2 (o un agente externo), que implementa la comunicación FIPA, y los servicios ofrecidos.

2.4.6. Jason y J-Moise

Magentix2 ofrece de forma nativa soporte para ejecutar agentes Jason y organizaciones J-MOISE+. Estos frameworks han sido integrados en Magentix2. Además, los agentes Jason pueden beneficiarse de las comunicaciones

[8] Véase sección 2.2

seguras, las facilidades de trazas y los mecanismos de seguridad ofrecidos por Magentix2.

2.4.7. Desarrollo

La distribución de Magentix2 incluye ejemplos del uso de agentes con la tecnología que Magentix2 ofrece. Hay ejemplos de agentes básicos, agentes conversacionales, del framework de THOMAS, seguridad, Jason y el uso de trazas.

Si se desea compilar un ejemplo de Magentix2 o crear un proyecto nuevo de Magentix se necesita incluir la librería en la ruta de las librerías de compilación de Java. Esta librería está en la carpeta *lib* dentro de la carpeta de instalación de Magentix2.

2.4.8. Instalación

Magentix2 está disponible para su descarga en la sección de descargas de la página web[9] del proyecto. La aplicación de la instalación ofrece interfaz gráfica, la cual guía al usuario a través del proceso de instalación. Es posible hacer una instalación completa o personalizada. Una instalación completa instala todo los componentes de Magentix2 necesarios para ejecutarlo correctamente, mientras que la instalación personalizada permite elegir qué componentes desea instalar.

[9] http://www.gti-ia.upv.es/sma/tools/magentix2/index.php

Capítulo 3

Dinamicidad en THOMAS/Magentix2

3.1. Estudio de la dinamicidad

Para modelar Organizaciones Virtuales dinámicas y abiertas es necesario tener una infraestructura diferente. [10] Esta infraestructura debe usar tecnología de agentes en el proceso de desarrollo y aplicar descomposición y abstracción. THOMAS y Magentix2, como hemos visto en el capítulo anterior, ofrece todo lo necesario para ofrecer dinamicidad en Organizaciones Virtuales. Esta arquitectura está formada por tres principales componentes que ofrecen dinamicidad.

- **Service Facilitator** (SF), ofrece un lugar donde las entidades autónomas pueden registrar la descripción de los servicios como una entrada de directorio.
- **Organization Management Service** (OMS), es responsable de especificar y administrar los componentes estructurales (roles, unidades y normas) y los componentes activos (agentes que están participando y los roles que tienen, unidades que están activas en cada momento, etc.).
- **Platform Kernel** (PK), es Magentix2 y ofrece los servicios básico de una plataforma de Sistema Multiagente e incorpora mecanismos de transporte de mensajes que facilitan la interacción entre los agentes.

Las ventajas de usar estos componentes para un sistema abierto, dinámico y descentralizado son:

- Permite dinamicidad: se pueden crear nuevas organizaciones en tiempo de ejecución, permitiendo el desarrollo del Sistema Multiagente emergiendo o cambiando dinámicamente. Además, las organizaciones también pueden destruirse cuando su objetivo es alcanzado.
- Mejora la forma de los comportamientos emergentes, como la composición de servicios emergentes: nuevos, relevantes y complejos servicios se pueden crear en tiempo de ejecución usando los servicios ya existentes.
- Mejora las técnicas de localización y composición de servicios: las entidades pueden publicar los servicios que ellas requieran (no solo los que ofrecen), debido a la dinamicidad de un sistema abierto, cuando una entidad llega al sistema y descubre que es posible ofrecer un servicio que se necesita, lo registra y lo ofrece.

- Permite expresar control normativo: el OMS es el encargado de controlar los derechos de cada rol. También almacena todas las normas definidas en el sistema y ofrece servicios para borrar y añadir normas.

Todos los métodos que afectan la creación de organizaciones, gestión de roles, registro de servicios y control de normas que tiene THOMAS han sido explicados en la sección 2.3. Es por ello que consideramos que todos estos servicios permiten dinamicidad en las Organizaciones Virtuales. Para comprobarlo, en el siguiente punto vamos a explicar el caso de estudios que hemos desarrollado sobre un Mercado de Turismo donde se han descrito varios escenario que nos presentarán las distintas posibilidades que soporta THOMAS/Magentix2.

3.2. Caso de estudio

El caso de estudio [10] que hemos desarrollado está enfocado en el sector del turismo. La idea es crear un sitio donde distintos agentes puedan crear sus agencias de viajes y puedan ofrecer sus servicios. Además, permita la entrada y salida de agentes que puedan buscar servicios, invocarlos y dejar el sistema, si así lo desean.

Con este ejemplo comprobaremos la entrada y la salida de agentes dentro de una organización, la creación de organizaciones, agencias, borrarlas, crear roles y borrarlos así como registrar distintos servicios y borrarlos. Todos ellos se harán en tiempo de ejecución, permitiendonos verificar la dinamicidad de THOMAS/Magentix2. Para ello hemos creado distintos escenarios para representar las distintas posibilidades que se nos puedan presentar. Además, hemos comprobado alguno de los escenarios usando agentes externos a la plataforma, concretamente agentes JADE, con lo que hemos verificando las propiedades de THOMAS/Magentix2 desde el exterior de la plataforma.

A continuación, describiremos la estructura (roles y unidades) que hemos usado y los servicios que aparecerán en el sistema. Después, presentaremos los distintos agentes que habrán el sistema, concretamente los agentes Magentix2 `GodAgent`, `ManagerAgency1`, `ManagerAgency2`, `ProviderAgent1` y `ClientAgent` y los agentes JADE `AgencyManagerJADE` y `ClientAgentJADE`. Y por último, presentaremos los distintos escenarios que hemos considerado.

3.2.1. Estructura

En esta sección vamos a presentar la estructura (roles, unidad) y servicios que habrá en el sistema. Para ello usaremos la notación definida en la metodología **GORMAS** [14].

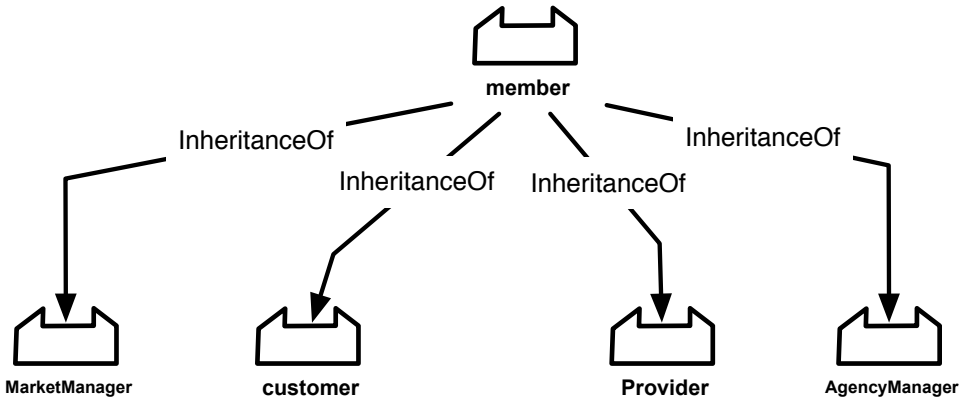

Figura 3.1: Esquema de las roles que hay en el sistema.

Roles y unidades

En el sistema tenemos los siguientes roles: `MarketManager`, `Provider`, `AgencyManager` y `customer`. En la figura 3.1 se pueden apreciar todos ellos.

- **Member**: Este rol es el rol por defecto en toda las unidades. Representa únicamente que un agente pertenece a una unidad. No tiene ninguna característica o funcionalidad adicional.
- **MarketManager**: Es el rol que desempeña el agente `GodAgent` y sólo él puede jugarlo. Este rol representa al director del Mercado de Turismo y solo el puede ofrecer los servicios de entrada al Mercado (`NewAgency`, `NewClient` y `NewProvider`).
- **Customer**: Este rol lo adquieren aquellos agentes que entran al sistema invocando el servicio `NewClient`. Como su nombre indica, son los clientes, por tanto, representa a aquellos agentes que quieren visitar el Mercado de Turismo para buscar y usar los servicios ofrecidos en él.
- **AgencyManager**: Este rol representa al director de una agencia de turismo. Está presente en cada una de las agencia que se creen en el Mercado, y lo adquiere un agente dentro de una agencia, cuando invoca el servicio `NewAgency`.
- **Provider**: Es el rol que adquiere un agente que invoca el servicio `NewProvider` porque quiere ofrecer algún servicio en el Mercado.

Las unidades que se crean en el sistema son `TourismMarket`, `Agency1` y `Agency2`. En la figura 3.2 se pueden ver.

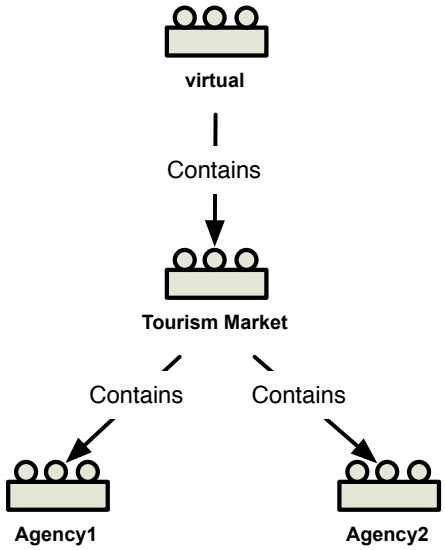

Figura 3.2: Esquema de las unidades que hay en el sistema.

- **Virtual**: Es la organización inicial que hay en el sistema. Y todos los agentes tiene que pertenecer a ella.

- **TourismMarket**: Es la organización principal de nuestro caso de estudio. Es el Mercado de Turismo, en ella se encuentra todas las agencias, clientes y proveedores.

- **Agency1**: Es la primera agencia que habrá en el sistema y estará en la situación inicial antes de ejecutar cada uno de los escenarios. En ella estará el agente `AgencyManager1`, ofrecerá el servicio de `SearchCheapHotel` y posteriormente necesitará el servicio `SearchCheapTrain` que será ofrecido por el agente `ProviderAgent1` el cual entrará en la unidad para ofrecerlo.

- **Agency2**: Es otra agencia de turismo que se creará de forma dinámica, concretamente es uno de los escenarios propuestos. Está agencia la creará el agente `AgencyManager2` y ofrecerá el servicio `SearchCheapFlight`

Finalmente la figura 3.3 muestra una visión general de qué unidades hay y qué roles hay en cada unidad.

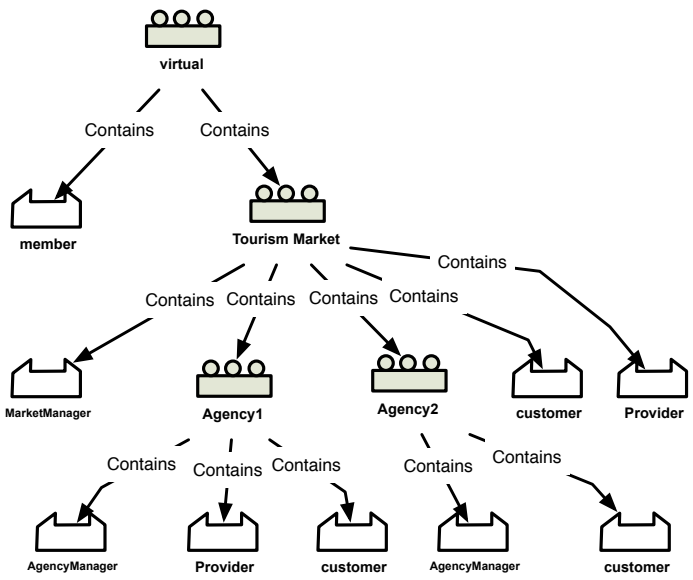

Figura 3.3: Esquema de la estructura del sistema.

Servicios

Los servicios ofrecidos por el rol `MarketManager` permiten la entrada al Mercado de Turismo y son los siguientes.

- ***NewAgency***: Servicio que crea la infraestructura necesaria (roles y unidades) para el agente que solicita crear una agencia dentro del Mercado de Turismo. Devuelve al agente la unidad y rol creado para su nueva agencia dentro de la unidad `MarketManager`.
- ***NewClient***: Servicio que valida la entrada de un agente cliente al Mercado de Turismo. Devuelve al agente el rol y la unidad, si se le permite entrar, que tiene que adquirir para entrar en el mercado.
- ***NewProvider***: Servicio que valida la entrada de un agente que quiera ofrecer un servicio dentro del Mercado de Turismo. Devuelve la unidad y el rol que tiene que adquirir para entrar dentro del mercado.

El resto de servicios son los que ofrecen las distintas agencias o el proveedor y son los siguientes.

- ***SearchCheapHotel***. Servicio que es ofrecido por el rol `AgencyManager` de la organización `Agency1`. El sistema de búsqueda de hotel que implementa es muy básico. Se le pasa como argumento, ciudad, país, categoría y rango de fechas y nos devuelve el nombre de un hotel.

- ***SearchCheapFlight***. Es ofrecido por el rol `AgencyManager` de la organización `Agency2`. El sistema de búqueda de vuelos que implementa es muy básico. Se le pasa como parámetros la ciudad origen, la ciudad destino y rango de fechas y el servicio devuelve la compañía del vuelo, el precio y el número de vuelo.

- ***SearchCheapTrain***. Lo ofrece el rol `Provider` dentro de la organización `Agency1`. Hace un búsqueda de trenes muy básica. Tiene como parámetros la ciudad destino y origen y el rango de fechas, el servicio devuelve el tipo de tren, la hora de salida y llegada, la duración del viaje y el precio.

3.2.2. Agentes

- **GodAgent**: Es el agente principal del Mercado de Turismo. Es el encargado de ofrecer los servicios de entrada al Mercado de Turismo *NewAgency*, *NewProvider* y *NewClient*, los cuales explicaremos mas abajo. Al entrar al sistema crear toda la infraestructura necesaria para poner en funcionamiento el Mercado de Turismo. Desempeña el rol de `MarketManager` dentro de la unidad `TourismMarket`.

- **ManagerAgency1**: Es el director de la agencia de turismo `Agency1` y ofrece el servicio `SearchCheapHotel`. Juega el rol de `AgencyManager` dentro de la unidad `Agency1`. Durante su ejecución decide publicar la descripción de un servicio, `SearchCheapTrain`, el cual, posteriormente, otro agente ofrecerá dentro de la agencia.

- **AgencyManagerJADE**: Es un agente JADE y realiza las mismas funciones que el agente Magentix2 `ManagerAgency1` pero desde la plataforma JADE.

- **ManagerAgency2**: Al igual que el agente `ManagerAgency1`, este agente es el director de una agencia de turismo, `Agency2`, y ofrece el servicio `SearchCheapHotel`

- **ProviderAgent1**: Es un agente Magentix2 que decide entrar en el Mercado de Turismo para ofrecer el servicio `SearchCheapTrain` el cual necesita la unidad `Agency1`. Desempeñará el rol `Provider` dentro de la unidad `Agency1`.

- **ClientAgent**: Es un agente que entrará al Mercado de Turismo y buscará, invocará un servicio y dejará el sistema.

- **ClientAgentJADE**: Es un agente JADE que realiza la misma función que el agente `ClientAgent`.

3.2.3. Escenarios

Para comprobar la dinamicidad de la plataforma, los distintos servicios y la especificación de THOMAS/Magentix2 hemos diseñado distintos escenarios que nos permitan representar las distintas posibilidades que se nos pueden presentar en un sistema dinámico.

La situación de inicial del sistema será la que tiene ya creada la unidad principal `TourismMarket`. Dentro de ella habrá creado un rol `MarketMananger` el cual desempeñará el agente `GodAgent`. Este agente, a su vez, ofrecerá tres servicios (`NewAgency`, `NewClient` y `NewProvider`) al exterior que permitan entrar a los agentes externos dentro del mercado, como comentamos anteriormente.

Además, en la situación inicial del Mercado de Turismo ya existirá una agencia, `Agency1`, en la cual habrá un agente, `AgencyManager1`, que desempeñará el rol de `AgencyManager` dentro de `Agency1` y ofrecerá el servicio `SearchCheapHotel`.

Una visión más clara de la situación inicial es la que muestra la figura 3.4. A partir de esta situación, empezaremos a ejecutar (como veremos en la siguiente sección) los distintos escenarios, como puede ser crear o borrar nuevas organizaciones o roles, adquirir o dejar roles, etc, y todo ello en tiempo de ejecución.

- **Crear Agencia**. El agente `AgencyManager2` invocando al servcio `New Agency` creará una nueva agencia, `Agency2` y ofrecerá el servicio `Search CheapFlight`. De esta forma comprobaremos la creación de una organización de forma dinámica. También probaremos lo mismo, pero usando un agente JADE, `AgencyManagerJADE`, para comprobar el acceso a la plataforma por un agente externo.

- **Nuevo proveedor**. La agencia, `Agency1` necesita el servicio `Search CheapHotel` y lo publica. A continuación, el agente `ProviderAgent1` entra en el sistema invocando el servicio `NewProvider` y ofrece ese servicio. Comprobamos si los agentes pueden publicar la necesidad de algún nuevo servicios y si un nuevo agente, al llegar al sistema, es capaz de darse cuenta de esta necesidad y cubrirla.

- **Nuevo cliente**. Comprobamos si un agente puede interactuar con la organización, es decir, entrar, buscar el servicio que necesito, invocar el servicio y dejar la organización. Para ello, distintos cliente entrarán

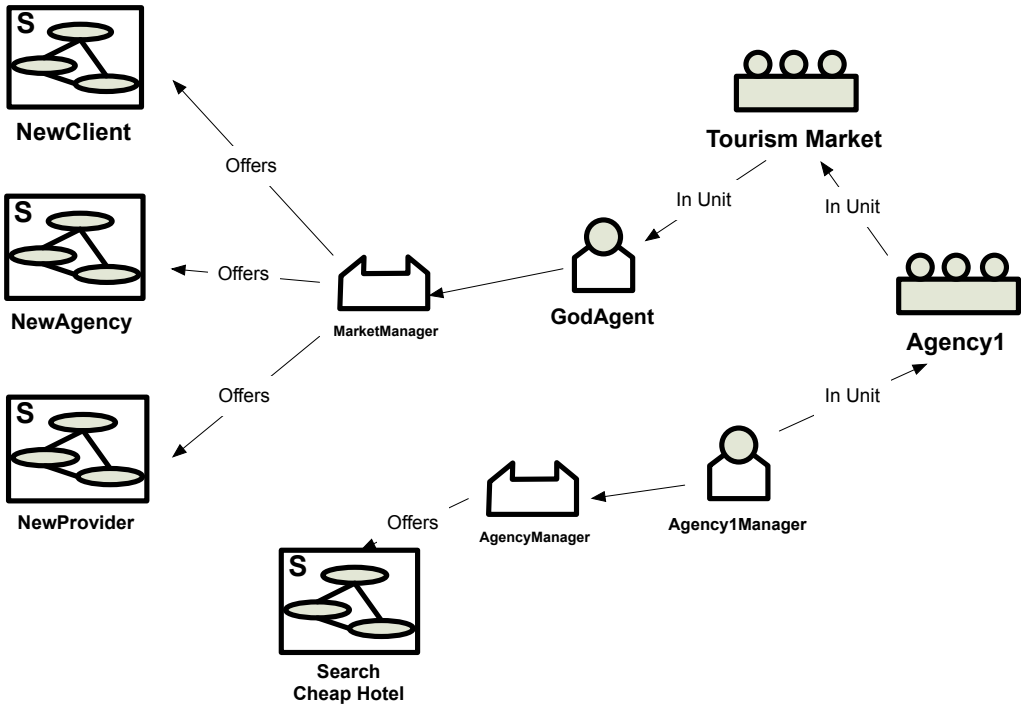

Figura 3.4: Situación inicial del Mercado de Turismo.

al sistema invocando el servicio `NewClient` y buscarán e invocarán los servicios ofrecidos. Serán agentes Magentix2, `ClientAgent`, y JADE, `ClientAgentJADE`.

- **Deshacer Agencia**. Comprobamos si un agente, de forma dinámica, puede dejar de ofrecer los servicios que ofrece y dejar el sistema debido a algún evento. El agente `AgencyManager2` decide deshacer la agencia `Agency2` y dejar el sistema.
- **Proveedor deja de ofrecer servicio**. El agente `ProviderAgent1` decide dejar de ofrecer el servicio `SearchCheapTrain` y dejar el sistema. De esta forma comprobamos que un agente que estaba ofreciendo un servicio pueda dejar de ofrecerlo pero que siga manteniendose la necesidad de ese servicio por parte de esa organización.
- **Deshacer Mercado de Turismo**. En este escenario comprobamos que se puedan eliminar organizaciones y roles que han sido creados previamente y de forma dinámica, es decir, que el agente deshace toda la infraestructura que ha creado antes de dejar el sistema. El agente `GodAgent` debido a algún evento, decide deshacer el Mercado de Turismo.

3.3. Pruebas

En esta sección vamos a explicar que software necesitamos tener instalado para poder ejecutar los distintos escenarios. Después, explicaremos, más detalladamente, en que consiste cada escenario y como ejecutarlos y que resultados hemos obtenido.

3.3.1. Requisitos iniciales

Para ejecutar el caso de estudio se necesita tener instalado el siguiente software:

- `MYSQL v 5.2.29`. Es el sistema de gestión de bases de datos del framework de THOMAS donde se guarda toda la información sobre las organizaciones, roles, agentes en el sistema, descripción e implementación de servicios, etc.
- `JADE v 3.7`. Plataforma de Sistemas Multiagente. Se necesita para ejecutar los agentes JADE que usamos para comprobar la interacción entra Magentix2 y otras plataformas.

- `Apache Tomcat v 5.5.31`. Es el servidor de servicios usado para desplegar los servicios que usamos, tanto los de THOMAS, como los que hemos creado nosotros.

- `Plataforma THOMAS/Magentix2`. La plataforma que queremos comprobar su dinamicidad y testear.

- `Servicios del caso de estudio`. Son los servicios implementados para el caso de estudio. Son `TourismMarket`, `SearchCheapHotel`, `Search CheapFlight` y `SearchCheapTrain`, los cuales hay que desplegar en Tomcat.

- `Agentes del caso de estudio`. Hay que tener descargado el código de los distintos agentes que se han implementado para ejecutar los escenarios propuestos.

3.3.2. Ejecución de los distintos escenarios

A continuación vamos a detallar como ejecutar los distintos escenarios y mostrar el resultado obtenido en cada uno de ellos.

Situación inicial

Para crear la situación inicial en la que comenzar a ejecutar los distintos escenarios se necesita tener en ejecución el servidor de bases de datos MYSQL y Tomcat con los servicios antes mencionados.

Dado que la prueba consiste en seguir cierto orden en la ejecución de comandos, no se puede automatizar.

Los pasos iniciales para reproducir la situación inicial son los siguientes:

- Reiniciar el servidor Tomcat. Con esto nos aseguramos de que los servicios web están *limpios*. Este paso previo no se hace debido a este test, se hace debido al diseño de los servicios de Thomas. Es el mejor modo de asegurarnos que la salida será la misma a la esperada, si todo va bien.

- Ejecutaremos el lanzador `RunMainOrganization` el cual lanza los agentes `GodAgent` y `ManagerAgency1` que realizan las siguientes acciones cada uno. El agente `GodAgent`:

    1. Entra en THOMAS, es decir, adquiere el rol "`member`"dentro de la organización "`virtual`".

2. Crea la organización "`TourismMarket`"
3. Crea el rol "`MarketManager`"dentro de la organización "`Tourism Market`".
4. Adquiere el rol "`MarketManager`"dentro de la organización "`Tourism Market`". Convirtiendose en el director de esta organización.
5. Crea el rol "`providerz`" "`customer`"dentro de organización "`Tourism Market`".
6. Añade la descripción y la implementación del servicio `NewClient`, `NewAgency` y `NewProvider`.
7. Se queda esperando que alguien invoque sus servicios.

Y el agente `ManagerAgency1` realiza las siguientes acciones:

1. Entra en THOMAS aquiriendo el rol "`member`"dentro de la organización "`virtual`".
2. Busca el servicio `NewAgency` y obtiene la descripción del servicio.
3. Adquiere el rol que le indica la descripción del servicio para invocarlo.
4. Obtiene la lista de proveedores a través de la implementación del servicio.
5. Invoca el servicio `NewAgency`.
6. Adquiere el rol que le ha devuelto la invocación del servicio para entrar en la agencia que se le ha creado dentro de la organización "`TourismMarket`".
7. Registra la descripción e implementación del servicio `SearchCheapHotel`.
8. Se queda esperando hasta que alguien invoque su servicio.

Al ejecutar el lanzador la salida que obtenemos por terminal es la que se muestra a continuación.

```
[ManagerAgency1] Entrar en THOMAS: Ok

[GodAgent]Entrar en THOMAS: Ok

[GodAgent]Register Unit TourismMarket result: Ok

[GodAgent]Register Role member result: Ok

[GodAgent]Register Role MarketManager result: Ok
```

```
[GodAgent]Acquire Role MarketManager result: Ok

[GodAgent]Register Role Payee result: Ok

[GodAgent]Register Role provider result: Ok

[GodAgent]Register Role customer result: Ok

[GodAgent] Register NewClientProfile result: 150

[GodAgent] Register NewClientProcess result:
150@144-qpid://GodAgent@localhost:8080

[GodAgent] Register NewAgencyProfile result: 151

[GodAgent] Register NewAgencyProcess result:
151@145-qpid://GodAgent@localhost:8080

[ManagerAgency1] Añado mi AID y el nombre de la agencia a crear
 como parametro: qpid://ManagerAgency1@localhost:8080 , Agency1

[ManagerAgency1] Agente que ofrece el servicio NewAgency: GodAgent

[ManagerAgency1] Ejecutamos el servicio.

[GodAgent] He recibido una petición de mi servicio NewAgency
[GodAgent] AGREE

[GodAgent] Executing... [Agency1, qpid://ManagerAgency1@localhost:8080]

[GodAgent] NewAgency: Registro de la nueva unidad, Agency1: Ok
[GodAgent] NewAgency: Registro del nuevo rol AgencyManager: Ok
[GodAgent] NewAgency: adquiere el rol el agente creador de la Agencia:
{http://localhost:8080/TourismMarket/owl/owls/
NewAgencyProcess.owl#NameUnit=Agency1,
http://localhost:8080/TourismMarket/owl/owls/
NewAgencyProcess.owl#AgentID=
qpid://ManagerAgency1@localhost:8080}
```

```
[GodAgent] NewAgency: Registro del nuevo rol Provider: Ok
[GodAgent]Creating Inform Message...
[ManagerAgency1] Valores devueltos.
      NewRol = AgencyManager
      ErrorValue = Ok!

[ManagerAgency1] Entrar en la Agencia Agency1 con el
rol AgencyManager: Ok

[GodAgent] Register NewProviderProfile result: 152

[GodAgent] Register NewProviderProcess result:
152@146-qpid://GodAgent@localhost:8080

[ManagerAgency1]Register SearchCheapHotel Profile result: 153

[ManagerAgency1] Register SearchCheapHotel Process result:
153@147-qpid://ManagerAgency1@localhost:8080
```

Al acabar de ejecutarse este *script*, conceptualmente, tendremos lo que muestra la figura 3.4. A partir de este momento podemos empezar a ejecutar los distintos escenarios.

Crear Agencia

Suponemos que una nueva agencia de viajes (`Agency2`) quiere entrar en el Mercado de Turismo y ofrecer un nuevo servicio (`SearchCheapFlight`) e intentar competir con el resto de agencias. Esta nueva agencia la creará un nuevo agente, `Agency2Manager`, el cual ofrecerá este nueve servicio de turismo. La figura 3.5 presenta los principales componentes de la nueva organización.

El primer paso consiste en registrar la nueva unidad. Para ello, la entidad, `Agency2Manager`, invoca el servicio `NewAgency` que ofrece la organización principal `TourismMarket`. Esta invocación conlleva un invocación interna del servicio *registerUnit* del OMS. Cuando la nueva unidad organizacional está creada, se podrán crear nuevos roles y unidades usando los servicios *registerRole* y *registerNorm* respectivamente.

En segundo lugar, el agente `Agency2Manager` puede entrar en la organización invocando el servicio correspondiente, *acquireRol*, ofrecido por el OMS. Asumiendo que todas las invocaciones se realizan correctamente, el agente habría adquirido el rol y la unidad que el servicio le habría respondido.

El último paso consiste en publicar el nuevo servicio en el SF. Para ello, el agente debe hacer el registro del servicio en el SF por el mismo. Para hacerlo,

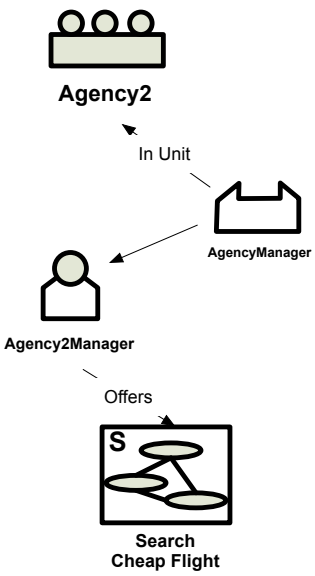

Figura 3.5: Muestra los componentes de la nueva unidad Agency2.

invocará el servicio *registerProfile* (para registrar la descripción general del servicio) y *registerProcess* (para registrar como invocar el servicio). Después de esto, la configuración del sistema habrá cambiado. La figura 3.6 recoge la nueva situación, donde una nueva organización ha sido añadida de forma dinámica.

Este escenario se ha implementado para que se pueda ejecutar con un agente de Magentix2 o con un agente de JADE. En resumen, el agente `ManagerAgency2` o `AgencyManagerJADE` realiza las siguientes acciones:

1. Entra en THOMAS aquiriendo el rol "`member`"dentro de la organización "`virtual`".

2. Busca el servicio `NewAgency` y obtiene la descripción del servicio.

3. Adquiere el rol que le indica la descripción del servicio para invocarlo.

4. Obtiene la lista de proveedores a través de la implementación del servicio.

5. Invoca el servicio `NewAgency`.

6. Adquiere el rol que le ha devuelto la invocación del servicio para entrar en la agencia que se le ha creado dentro de la organización "`TourismMarket`".

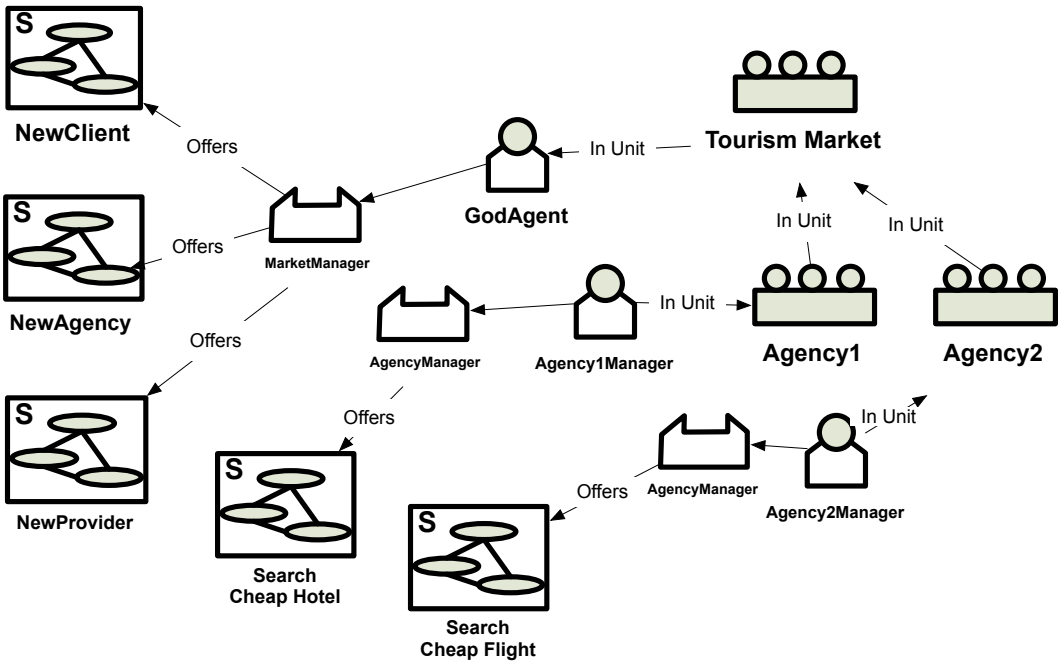

Figura 3.6: Muestra la nueva situación del sistema después de haberse creado la nueva unidad Agency2.

7. Registra la descripción e implementación del servicio `SearchCheapFlight`.

8. Se queda esperando hasta que alguien invoque su servicio.

Para ejecutarlo con un agente de Magentix2 ejecutaremos el lanzador `RunAgency2` obteniendo por consola el siguiente resultado:

```
[ManagerAgency2] Entrar en THOMAS: Ok

[ManagerAgency2] Añado mi AID y el nombre de la agencia a crear
 como parametro: qpid://ManagerAgency2@localhost:8080 , Agency2

[ManagerAgency2] Agente que ofrece el servicio NewAgency: GodAgent

[ManagerAgency2] Ejecutamos el servicio.

[ManagerAgency2] Valores devueltos.
      NewRol = AgencyManager
      ErrorValue = Ok!

[ManagerAgency2] Entrar en la Agencia Agency2 con el rol AgencyManager: Ok

[ManagerAgency2]Register SearchCheapFlight Profile result: 154

[ManagerAgency2] Register SearchCheapFlgiht Process result:
154@147-qpid://ManagerAgency2@localhost:8080
```

Si queremos ejecutarlo con un agente JADE introducimos el agente `AgencyManagerJADE` en la plataforma JADE. El resultado obtenido en la consola de JADE será:

```
I'm the ManagerAgency@bowser:1099/JADE Agent!!!!
ManagerAgency: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 AcquireRoleProcess.owl RoleID=member UnitID=virtual
 AgentID=ManagerAgency@bowser:1099/JADE"
 :protocol  fipa-request
)
```

```
ManagerAgency: SearchService =(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 SearchServiceProcess.owl SearchServiceInputServicePurpose=NewAgency"
 :protocol  fipa-request
)
ManagerAgency: OOH! OMS Has agreed to excute the service AcquireRol!
ManagerAgency: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
ManagerAgency: OOH! SF Has agreed to excute the service SearchService!
ManagerAgency: GetProfile =(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 GetProfileProcess.owl GetProfileInputServiceID=12"
 :protocol  fipa-request
)
ManagerAgency: SF SearchService IDProfile = 12
ManagerAgency: OOH! SF Has agreed to excute the service GerProfile!
ManagerAgency: SF GetProfile URL = http://localhost:8080/TourismMarket/
owl/owls/NewAgencyProfile.owl
ManagerAgency: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 AcquireRoleProcess.owl RoleID=member UnitID=tourismmarket
 AgentID=ManagerAgency@bowser:1099/JADE"
 :protocol  fipa-request
)
ManagerAgency: GetProcess =(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
```

```
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 GetProcessProcess.owl GetProcessInputServiceID=12"
 :protocol  fipa-request
)
ManagerAgency: OOH! OMS Has agreed to excute the service AcquireRol!
ManagerAgency: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
ManagerAgency: OOH! SF Has agreed to excute the service GetProcess!
ManagerAgency:SF has informed me of the status of my request. They said :
GetProcessProcess={http://localhost:8080/sfservices/SFservices/owl/owls/
GetProcessProcess.owl#GetProcessOutputProcessList=12@12-GodAgent
http://localhost:8080/TourismMarket/owl/owls/NewAgencyProcess.owl,
http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
#GetProcessOutputServiceStatus=1}
ManagerAgency: SF GetProcess Provider = GodAgent
[ManagerAgency]Sms to send: (REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name GodAgent@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/TourismMarket/owl/owls/NewAgencyProcess.owl
 AgentID=ManagerAgency@bowser:1099/JADE NameUnit=Agency2"
 :protocol  fipa-request
)
[ManagerAgency]Sending...
ManagerAgency: OOH! GodAgent Has agreed to excute
the service GenericService!
ManagerAgency:GodAgent has informed me of the status of my request.
They said: NewAgencyProcess={http://localhost:8080/
TourismMarket/owl/owls/NewAgencyProcess.owl#NewRol=
AgencyManager, http://localhost:8080/TourismMarket/owl/owls/
NewAgencyProcess.owl #ErrorValue=Ok!}

ManagerAgency: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 AcquireRoleProcess.owl RoleID=AgencyManager UnitID=Agency2
 AgentID=ManagerAgency@bowser:1099/JADE"
 :protocol  fipa-request
)
```

```
ManagerAgency: SearchService =(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 SearchServiceProcess.owl SearchServiceInputServicePurpose=SearchCheapFlight"
 :protocol  fipa-request
)
ManagerAgency: OOH! OMS Has agreed to excute the service AcquireRol!
ManagerAgency: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
ManagerAgency: OOH! SF Has agreed to excute the service SearchService!
(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 RegisterProfileProcess.owl RegisterProfileInputServiceGoal=
 RegisterProfileInputServiceProfile=http://localhost:8080/
 SearchCheapFlight/owl/owls/SearchCheapFlightProfile.owl#SearchCheapFlight"
 :protocol  fipa-request
)
ManagerAgency: SF SearchService IDProfile =
ManagerAgency: OOH! SF Has agreed to excute the RegisterProfile!
ManagerAgency:SF(INFORM
 :sender  ( agent-identifier :name SF@bowser:8081
 :addresses (sequence http://bowser.dsic.upv.es:8081 ))
 :receiver  (set ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778 )) )
 :content  "RegisterProfileProcess={http://localhost:8080/sfservices/
 SFservices/owl/owls/RegisterProfileProcess.owl#
 RegisterProfileOutputServiceStatus=1,
 http://localhost:8080/sfservices/SFservices/owl/owls/
 RegisterProfileProcess.owl#RegisterProfileOutputServiceID=15}"
 :protocol  fipa-request
 :conversation-id  C29291718_1290456334963 )
```

```
(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 RegisterProcessProcess.owl RegisterProcessInputServiceModel=
 http://localhost:8080/SearchCheapFlight/owl/owls/
 SearchCheapFlightProcess.owl#SearchCheapFlightProcess
 RegisterProcessInputServiceID=15"
 :protocol  fipa-request
)
ManagerAgency:SF RegisterProfile has returned URL = 15
ManagerAgency: OOH! SF Has agreed to excute the RegisterProcess!
ManagerAgency:SF(INFORM
 :sender  ( agent-identifier :name SF@bowser:8081
 :addresses (sequence http://bowser.dsic.upv.es:8081 ))
 :receiver  (set ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778 )) )
 :content  "RegisterProcessProcess={http://localhost:8080/sfservices/
 SFservices/owl/owls/RegisterProcessProcess.owl#
 RegisterProcessOutputServiceModelID=
 5@15-ManagerAgency@bowser:1099/JADE,
 http://localhost:8080/sfservices/SFservices/owl/owls/
 RegisterProcessProcess.owl#RegisterProcessOutputServiceStatus=1}"
 :protocol  fipa-request
 :conversation-id  C27253500_1290456338713 )
ManagerAgency: SF RegisterProcess has returned =
15@15-ManagerAgency@bowser:1099/JADE
```

Nuevo proveedor

Una de las principales características de esta plataforma es la posibilidad de que un agente u organización pueda publicar la necesidad de algún servicio. De esta manera y, gracias a la dinamicidad de las entidades en THOMAS, cuando una nueva entidad llega al sistema puede darse cuenta de las necesidades que hay en el sistema.

En el siguiente escenario, el agente `Agency1Manager` ha decidido añadir un servicio de búsqueda de trenes, `SearchCheapTrain`, ya que se ha dado cuenta que ese servicio ha sido solicitado muchas veces, pero nadie lo estaba ofreciendo. Para ello, ha añadido la descripción del servicio al SF usando el servicio `registerProfile`.

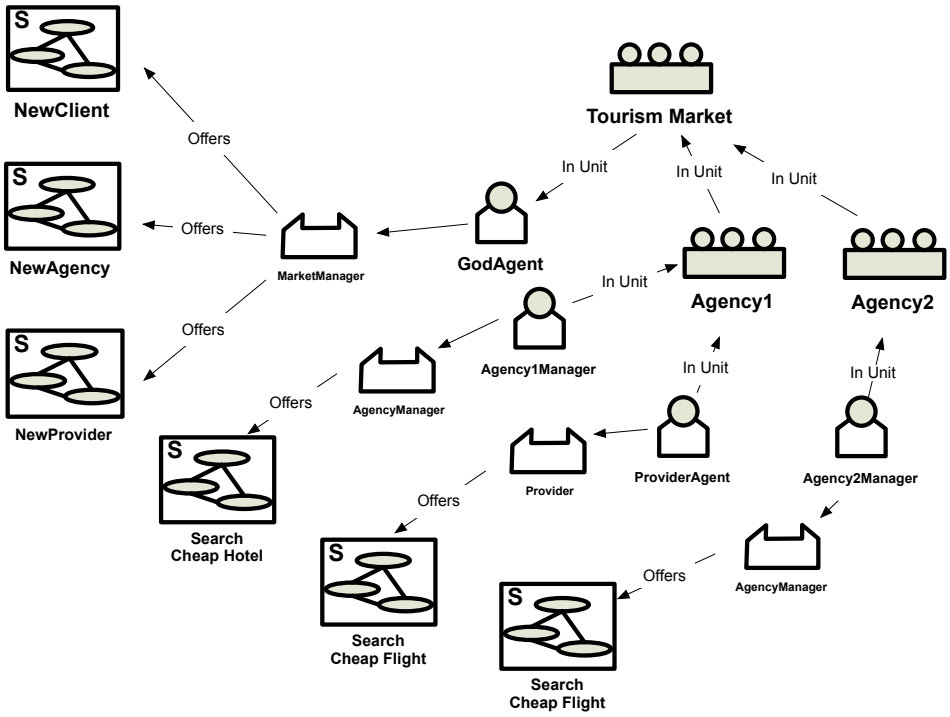

Figura 3.7: Muestra el nuevo agente proveedor dentro del sistema.

Después de esto, el agente `ProviderAgent` llega al sistema y lo primero que hace es entrar en la unidad `Virtual` (organización que existe por defecto en THOMAS). A continuación, invoca el servicio `NewProvider` y adquiere el rol y la unidad que necesita para entrar en la unidad `TourismMarket`. El agente busca en el SF (invocando el servicio `searchService`) todos los posibles servicio que el podría ofrecer y encuentra que alguien necesita el servicio `SearchCheapTrain` el cual el puede ofrecer. Con el `ID` del servicio que le ha devuelto el servicio `searchService` obtiene la descripción del servicio donde se especifica el rol y la unidad que tiene que adquirir para ofrecer ese servicio y la lista de proveedores que lo ofrecen. Así que adquiere el rol dentro la unidad correspondiente y registra el servicio en el SF. La figura 3.7 detalla los elementos añadidos al sistema en este escenario y la situación en la que se encontraría el sistema.

Para ejecutar este escenario tendremos que ir a la interfaz gráfica de `Agency1` y hacer click en `Añadir profile` como muestra la figura 3.8

Al hacer click, en la consola aparecerá:

```
[ManagerAgency1] Register SCT  Profile result: 155
```

Figura 3.8: Interfaz gráfica del agente `ManagerAgency1`

Ahora, tendremos que introducir el agente `ProviderAgent`, que realiza las siguientes acciones:

1. Entra en THOMAS adquiriendo el rol "`member`"dentro de la organización "`virtual`".

2. Busca el servicio `NewProvider` y obtiene la descripción del servicio.

3. Adquiere el rol que le indica la descripción del servicio para invocarlo.

4. Obtiene la lista de proveedores a través de la implementación del servicio.

5. Invoca el servicio `NewAgency`.

6. Adquiere el rol que le ha devuelto la invocación del servicio para entrar en la agencia que se le ha creado dentro de la organización "`TourismMarket`".

7. Busca el servicio `SearchCheapTrain` y obtiene su descripción.

8. Adquiere el rol que le indica la descripción del servicio para poder ofrecer el servicio.

9. Registra la descripción e implementación del servicio `SearchCheapTrain`.

10. Se queda esperando hasta que alguien invoque su servicio.

Para introducir el agente en el sistema, ejecutaremos el lanzador `RunProvider` y obtendremos por consola el siguiente resultado:

```
[ProviderAgent1] Entrar en THOMAS: Ok

[ProviderAgent1] Anyado mi AID como parametro:
        qpid://ProviderAgent1@localhost:8080

[ProviderAgent1] Agente que ofrece el servicio NewProvider: GodAgent

[ProviderAgent1] Ejecutamos el servicio.

[ProviderAgent1] Valores devueltos.
      NewUnit = TourismMarket
      NewRol = provider
      ErrorValue = Ok!

[ProviderAgent1] Entrar en TourismMarket: Ok

[ProviderAgent1] Comprobamos si alguien quiere el servicio SearchCheapTrain

[ProviderAgent1] Alguien quiere el servicio SearchCheapTrain
y esta registrado en el SF con id: 155

[ProviderAgent1] Adquirir rol customer: Ok

[ProviderAgent1] Register SearchCheapTrain Process result:
155@148-qpid://ProviderAgent1@localhost:8080
```

Nuevo cliente

Un agente `ClientAgent` desea buscar y usar una serie de servicios dentro del Mercado de Turismo. Para ello, el agente invoca el servicio `NewClient` que ofrece `GodAgent` para entrar dentro del Mercado de Turismo. El servicio valida si el agente puede entrar y le devuelve el rol y unidad que tiene que adquirir para poder entrar dentro de la organización. Cuando ha adquirido el nuevo rol invoca el servicio `SearchService` para buscar los servicio `SearchCheapHotel`, `SearchCheapFlight` o `SearchCheapTrain`. Con el identificador de uno de los servicios obtiene la descripción (*Profile Description*) y la implementación (*Process Description*) asociado al servicio. En la descripción del servicio obtiene el rol que tiene que tener dentro de una unidad para poder invocar el servicio y en la implementación del servicio obtiene los parámetros de entrada y de salida del servicio y que agente lo ofrece. Finalmente el agente invoca el servicio, obtiene la información deseada y deja el

Mercado de Turismo invocando el servicio `leaveRol` del OMS.

Este escenario se ha ejecutado de dos formas distintas atendiendo al tipo de agente cliente, si es un agente Magentix2 o JADE. La primera es que un agente Magentix2 ejecuta un servicio ofrecido por un agente Magentix2 y luego ejecuta un servicio ofrecido por un agente JADE. La segunda forma es un agente JADE que ejecuta un servicio ofrecido por un agente Magentix2 y luego ejecuta un servicio ofrecido por un agente JADE. Ambos agentes realizan las siguientes acciones:

1. Entra en THOMAS adquiriendo el rol "`member`"dentro de la organización "`virtual`".

2. Busca el servicio `NewClient` y obtiene la descripción del servicio.

3. Adquiere el rol que le indica la descripción del servicio para invocarlo.

4. Obtiene la lista de proveedores a través de la implementación del servicio.

5. Invoca el servicio `NewClient`.

6. Adquiere el rol que le ha devuelto la invocación del servicio para entrar en la agencia que se le ha creado dentro de la organización "`TourismMarket`".

7. Busca el servicio que desea invocar, `SearchCheapHotel`, `SearchCheapFlight` o `SearchCheapTrain` y obtiene su descripción.

8. Adquiere el rol que le indica la descripción del servicio para poder invocarlo.

9. Obtiene la lista de proveedores a través de la implementación del servicio.

10. Invoca el servicio que haya buscado.

11. Deja el sistema, dejando de jugar el rol "`member`"dentro de la organización "`virtual`".

En primer lugar, vamos a ejecutar tres agentes Magentix2 donde cada uno ejecutará un servicio ofrecido por un agente Magentix2. Para ello ejecutaremos el lanzador `RunClient` y obtendremos el siguiente resultado:

```
[ClientAgent1] Entrar en THOMAS: Ok

[ClientAgent2] Entrar en THOMAS: Ok

[ClientAgent3] Entrar en THOMAS: Ok

[ClientAgent1] Añado mi AID como parametro:
        qpid://ClientAgent1@localhost:8080

[ClientAgent1] Agente que ofrece el servicio NewClient: GodAgent

[ClientAgent1] Ejecutamos el servicio.

[ClientAgent1] Valores devueltos.
      NewUnit = TourismMarket
      NewRol = customer
      ErrorValue = Ok!
[ClientAgent1] Entrar en TourismMarket: Ok

[ClientAgent2] Añado mi AID como parametro:
        qpid://ClientAgent2@localhost:8080

[ClientAgent2] Agente que ofrece el servicio NewClient: GodAgent

[ClientAgent2] Ejecutamos el servicio.

[ClientAgent2] Valores devueltos.
      NewUnit = TourismMarket
      NewRol = customer
      ErrorValue = Ok!

[ClientAgent2] Entrar en TourismMarket: Ok

[ClientAgent3] Añado mi AID como parametro:
        qpid://ClientAgent3@localhost:8080

[ClientAgent3] Agente que ofrece el servicio NewClient: GodAgent

[ClientAgent3] Ejecutamos el servicio.

[ClientAgent3] Valores devueltos.
      NewUnit = TourismMarket
      NewRol = customer
      ErrorValue = Ok!
```

```
[ClientAgent3] Entrar en TourismMarket: Ok

[ClientAgent1] Añadimos los parametros de entrada:...

[ClientAgent1] Agente que ofrece el servicio SearchCheapFlight: ManagerAgency2

[ClientAgent1] Valores devueltos.
      SearchCheapFlightOutputFlightCompany = Oceanic
      SearchCheapFlightOutputFlight = 815
      SearchCheapFlightOutputPrice = 750

[ClientAgent1] Vamos a salir...

[ClientAgent2] Añadimos los parametros de entrada:...

[ClientAgent2] Agente que ofrece el servicio SearchCheapTrain: ProviderAgent1

[ClientAgent1] Salir de THOMAS : Ok

[ClientAgent2] Valores devueltos.
      SearchCheapTrainOutputTrain = Alaris
      SearchCheapTrainOutputDeparture = 10:15
      SearchCheapTrainOutputArrival = 13:15
      SearchCheapTrainOutputDuration = 3:00
      SearchCheapTrainOutputPrice = 47

[ClientAgent2] Vamos a salir...

[ClientAgent2] Salir de THOMAS : Ok

[ClientAgent3] Añadimos los parametros de entrada: Categoria = 2 ,
Pais = Spain y Ciudad = Valencia

[ClientAgent3] Agente que ofrece el servicio SearchCheapHotel: ManagerAgency1

[ClientAgent3] Valores devueltos.
      SearchCheapHotelOutputHotelCompany = NH
      SearchCheapHotelOutputHotel = Hotel Puerta de Valencia

[ClientAgent3] Vamos a salir...

[ClientAgent3] Salir de THOMAS : Ok
```

En segundo lugar, ejecutaremos un agente Magentix2 cliente que ejecuta un servicio ofrecido por un agente en JADE. Para ello ejecutaremos el lanzador `RunClient` modificado[1], para que solo ejecute el servicio *SearchCheapFlight*, como se ha explicado anteriormente. El resultado obtenido será:

```
[ClientAgent1] Entrar en THOMAS: Ok

[ClientAgent1] Anyado mi AID como parametro:
qpid://ClientAgent1@localhost:8080

[ClientAgent1] Agente que ofrece el servicio NewClient: GodAgent

[ClientAgent1] Ejecutamos el servicio.

[ClientAgent1] Valores devueltos.
      NewUnit = TourismMarket
      NewRol = customer
      ErrorValue = Ok!
[ClientAgent1] Entrar en TourismMarket: Ok

[ClientAgent1] Anyadimos los parametros de entrada:...

[ClientAgent1] Agente que ofrece el servicio SearchCheapFlight:
 AgencyManagerJADE@bowser:1099/JADE

[ClientAgent1] Valores devueltos.
      SearchCheapFlightOutputFlightCompany = Oceanic
      SearchCheapFlightOutputFlight = 815
      SearchCheapFlightOutputPrice = 750?

[ClientAgent1] Vamos a salir...

[ClientAgent1] Salir de THOMAS: Ok
```

A continuación, ejecutaremos un agente en JADE que invocará un servicio ofrecido por un agente JADE. Para ello introducimos el agente `ClientAgentJADE` en la plataforma JADE obteniendo el siguiente resultado:

```
I'm the Client@bowser:1099/JADE Agent!!!!
```

[1]La modificación consiste en comentar la línea de la instrucción *servicios.add("SearchCheapTrain")* y cambiar la inicialización de la variable N por 1

```
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 AcquireRoleProcess.owl RoleID=member UnitID=virtual
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: SearchService =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 SearchServiceProcess.owl SearchServiceInputServicePurpose=NewClient"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service SearchService!
Client: GetProfile =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 GetProfileProcess.owl GetProfileInputServiceID=11"
 :protocol  fipa-request
)
Client: SF SearchService IDProfile = 11
Client: OOH! SF Has agreed to excute the service GerProfile!
Client: SF GetProfile URL =
http://localhost:8080/TourismMarket/owl/owls/NewClientProfile.owl
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
```

```
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 AcquireRoleProcess.owl RoleID=member UnitID=tourismmarket
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: GetProcess =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 GetProcessProcess.owl GetProcessInputServiceID=11"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service GetProcess!
Client:SF has informed me of the status of my request. They said :
GetProcessProcess={http://localhost:8080/sfservices/SFservices/owl/owls/
GetProcessProcess.owl#GetProcessOutputProcessList=11@11-GodAgent
http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl,
http://localhost:8080/sfservices/SFservices/owl/owls/
GetProcessProcess.owl#GetProcessOutputServiceStatus=1}
Client: SF GetProcess Provider = GodAgent
Client: Ejecutando servicio...
[Client]Sms to send: (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name GodAgent@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/TourismMarket/owl/owls/
 NewClientProcess.owl AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
[Client]Sending...
Client: OOH! GodAgent Has agreed to excute the service GenericService!
Client:GodAgent has informed me of the status of my request. They said :
NewClientProcess={http://localhost:8080/TourismMarket/owl/owls/
NewClientProcess.owl#NewRol=customer, http://localhost:8080/TourismMarket/
owl/owls/NewClientProcess.owl#NewUnit=TourismMarket, http://localhost:8080/
TourismMarket/owl/owls/NewClientProcess.owl#ErrorValue=Ok!}
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
```

```
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 AcquireRoleProcess.owl RoleID=customer UnitID=TourismMarket
  AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: SearchService =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 SearchServiceProcess.owl SearchServiceInputServicePurpose=
 SearchCheapFlight"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service SearchService!
Client: GetProfile =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 GetProfileProcess.owl GetProfileInputServiceID=15"
 :protocol  fipa-request
)
Client: SF SearchService IDProfile = 15
Client: OOH! SF Has agreed to excute the service GerProfile!
Client: SF GetProfile URL = http://localhost:8080/SearchCheapFlight/
owl/owls/SearchCheapFlightProfile.owl
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 AcquireRoleProcess.owl RoleID=customer UnitID=tourismmarket
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
```

```
Client: GetProcess =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 GetProcessProcess.owl GetProcessInputServiceID=15"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Error ErrorValue = Duplicate
Client: OOH! SF Has agreed to excute the service GetProcess!
Client:SF has informed me of the status of my request. They said :
GetProcessProcess={http://localhost:8080/sfservices/SFservices/
owl/owls/GetProcessProcess.owl#GetProcessOutputProcessList=
15@15-ManagerAgency@bowser:1099/JADE
http://localhost:8080/SearchCheapFlight/owl/owls/
SearchCheapFlightProcess.owl, http://localhost:8080/sfservices/
SFservices/owl/owls/GetProcessProcess.owl
#GetProcessOutputServiceStatus=1}
Client: SF GetProcess Provider =
        ManagerAgency@bowser:1099/JADE
Client: Ejecutando servicio...
[Client]Sms to send: (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name
         ManagerAgency@bowser:1099/JADE ) )
 :content  "http://localhost:8080/SearchCheapFlight/owl/owls/
 SearchCheapFlightProcess.owl
 SearchCheapFlightInputDepartingFrom=Sydney
 SearchCheapFlightInputGoingTo=Nueva York
 SearchCheapFlightInputDepartDate=22/10/00
 SearchCheapFlightInputReturnDate=10/11/00
 SearchCheapFlightInputNumberOfPassanger=2"
 :protocol  fipa-request
)
[Client]Sending...
ManagerAgency: Doc OWL-S: http://localhost:8080/
SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl
ManagerAgency: AGREE
```

```
ManagerAgency: Sending First message:(AGREE
 :receiver  (set ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc )) )
 :content  "SearchCheapFlightProcess=Agree"
 :reply-with  Client@bowser:1099/JADE1290456368635
 :protocol  fipa-request
 :conversation-id  C1142607_1290456368635 )
Client: OOH! ManagerAgency Has agreed to excute
        the service GenericService!
ManagerAgency: Doc OWL-S: http://localhost:8080/SearchCheapFlight/
owl/owls/SearchCheapFlightProcess.owl
ManagerAgency: Executing... [Sydney, 10/11/00, 22/10/00, 2, Nueva]
ManagerAgency: Values obtained... :
http://localhost:8080/SearchCheapFlight/owl/owls/
SearchCheapFlightProcess.owl #SearchCheapFlightOutputPrice=750,
http://localhost:8080/SearchCheapFlight/owl/owls/
SearchCheapFlightProcess.owl
#SearchCheapFlightOutputFlightCompany=Oceanic,
http://localhost:8080/SearchCheapFlight/owl/owls/
SearchCheapFlightProcess.owl#SearchCheapFlightOutputFlight=815
ManagerAgency: Creating inform message to send...
ManagerAgency: Before set message content...
Client:ManagerAgency has informed me of the status of my request.
They said : SearchCheapFlightProcess=http://localhost:8080/
SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl
#SearchCheapFlightOutputPrice=750,
http://localhost:8080/SearchCheapFlight/owl/owls/
SearchCheapFlightProcess.owl
#SearchCheapFlightOutputFlightCompany=Oceanic,
http://localhost:8080/SearchCheapFlight/owl/owls/
SearchCheapFlightProcess.owl
#SearchCheapFlightOutputFlight=815
[Client] Valores devueltos.
      SearchCheapFlightOutputFlightCompany = Oceanic
      SearchCheapFlightOutputFlight = 815
      SearchCheapFlightOutputPrice = 750
Client: Salir de THOMAS
Client: LeaveRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
```

```
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 LeaveRoleProcess.owl RoleID=member UnitID=virtual
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
```

Y, por último, ejecutaremos un agente cliente JADE que invocará un servicio ofrecido por un agente Magentix2. Para ellos volveremos a introducir el mismo agente, `ClientAgentJADE` invocando el servicio *SearchCheapHotel* que ofrece el agente `AgencyManagerJADE`. El resultado obtenido será:

```
I'm the Client@bowser:1099/JADE Agent!!!!
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 AcquireRoleProcess.owl RoleID=member UnitID=virtual
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: SearchService =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 SearchServiceProcess.owl SearchServiceInputServicePurpose=NewClient"
 :protocol  fipa-request
)
```

```
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service SearchService!
Client: GetProfile =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 GetProfileProcess.owl GetProfileInputServiceID=26"
 :protocol  fipa-request
)
Client: SF SearchService IDProfile = 26
Client: OOH! SF Has agreed to excute the service GerProfile!
Client: SF GetProfile URL = http://localhost:8080/TourismMarket/owl/owls/
NewClientProfile.owl
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 AcquireRoleProcess.owl RoleID=member UnitID=tourismmarket
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: GetProcess =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 GetProcessProcess.owl GetProcessInputServiceID=26"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service GetProcess!
Client:SF has informed me of the status of my request. They said :
GetProcessProcess=http://localhost:8080/sfservices/SFservices/owl/owls/
GetProcessProcess.owl#GetProcessOutputProcessList=26@25-GodAgent
http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl,
http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
#GetProcessOutputServiceStatus=1
```

```
Client: SF GetProcess Provider = GodAgent
Client: Ejecutando servicio...
[Client]Sms to send: (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name GodAgent@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
[Client]Sending...
Client: OOH! GodAgent Has agreed to excute the service GenericService!
Client:GodAgent has informed me of the status of my request. They said :
NewClientProcess=http://localhost:8080/TourismMarket/owl/owls/
NewClientProcess.owl#NewRol=customer,
http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl
#NewUnit=TourismMarket,
http://localhost:8080/TourismMarket/owl/owls/
NewClientProcess.owl#ErrorValue=Ok!
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 AcquireRoleProcess.owl RoleID=customer UnitID=TourismMarket
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: SearchService =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 SearchServiceProcess.owl
 SearchServiceInputServicePurpose=SearchCheapHotel"
 :protocol  fipa-request
)
```

```
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service SearchService!
Client: GetProfile =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 GetProfileProcess.owl GetProfileInputServiceID=29"
 :protocol  fipa-request
)
Client: SF SearchService IDProfile = 29
Client: OOH! SF Has agreed to excute the service GerProfile!
Client: SF GetProfile URL = http://localhost:8080/SearchCheapHotel/
owl/owls/SearchCheapHotelProfile.owl
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 AcquireRoleProcess.owl RoleID=customer UnitID=tourismmarket
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: GetProcess =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/
 GetProcessProcess.owl GetProcessInputServiceID=29"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Error ErrorValue = Duplicate
Client: OOH! SF Has agreed to excute the service GetProcess!
Client:SF has informed me of the status of my request. They said :
GetProcessProcess=http://localhost:8080/sfservices/SFservices/
owl/owls/GetProcessProcess.owl
#GetProcessOutputProcessList=29@28-ManagerAgency1
http://localhost:8080/SearchCheapHotel/owl/owls/
SearchCheapHotelProcess.owl,
http://localhost:8080/sfservices/SFservices/owl/owls/
GetProcessProcess.owl#GetProcessOutputServiceStatus=1
```

```
Client: SF GetProcess Provider = ManagerAgency1
Client: Ejecutando servicio...
[Client]Sms to send: (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name ManagerAgency1@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/SearchCheapHotel/owl/owls/
 SearchCheapHotelProcess.owl
 SearchCheapHotelInputCategory=categoria
 SearchCheapHotelInputCountry=España
 SearchCheapHotelInputCity=Valencia"
 :protocol  fipa-request
)
[Client]Sending...
Client: OOH! ManagerAgency1 Has agreed to excute
         the service GenericService!
Client:ManagerAgency1 has informed me of the status of my request.
They said : SearchCheapHotelProcess=
http://localhost:8080/SearchCheapHotel/owl/owls/
SearchCheapHotelProcess.owl
#SearchCheapHotelOutputHotel=Hotel Puerta de Valencia,
http://localhost:8080/SearchCheapHotel/owl/owls/
SearchCheapHotelProcess.owl
#SearchCheapHotelOutputHotelCompany=NH
[Client] Valores devueltos.
      SearchCheapHotelOutputHotelCompany = NH
      SearchCheapHotelOutputHotel = Hotel Puerta de Valencia
Client: Salir de THOMAS
Client: LeaveRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/
 LeaveRoleProcess.owl RoleID=member UnitID=virtual
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service LeaveRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
```

Deshacer Agencia

La unidad `Agency2` ha decidido disolverse debido a un evento como que la agencia no funcione como se esperaba, que el agente ha infringido una norma, que el agente no quiere continuar ofreciendo sus servicio, etc. El primer paso que el agente `Agency2Manager` debe hacer es borrar el servicio y después salir de THOMAS. Para borrar su servicio, `SearchCheapFlight`, `Agency2Manager` invoca el servicio `deregisterProcess` y `deregisterProfile` del *Service Facilitator* (SF). Como consecuencia de esto, ningún otro agente podrá encontrar este servicio ya que ha sido borrada la descripción y la implementación del mismo.

Después, el agente saldrá de la unidad `virtual` invocando el servicio `leaveRol` del OMS y finalmente habrá sido disuelta la unidad `Agency2` del Mercado de Turismo. Las acciones que realiza el agente serán las siguientes:

1. Elimina la implementación y la descripción del servicio que ofrece.
2. Sale de THOMAS, dejando el rol "`member`"de la organización "`virtual`"
3. Deja el sistema, dejando de jugar el rol "`member`"dentro de la organización "`virtual`".

Para ejecutar este escenario hay que hacer click en la interfaz gráfica de `Funciones de Agency2` en el botón `Deshacer` y obtendremos el siguiente resultado:

```
[ManagerAgency2] Eliminarme como provedor del servicio SearchCheapFlight:
154@147-qpid://ManagerAgency2@localhost:8080

[ManagerAgency2] Eliminar el profile del servicio SearchCheapFlight: 1

[ManagerAgency2] Salir de THOMAS: Ok
```

Proveedor deja de ofrecer servicio

De la misma forma que en la sección anterior, el agente `Agency2Manager` decide disolver la agencia debido a un evento, en este caso, el agente `ProviderAgent` decide dejar de ofrecer su servicio debido a un evento cualquiera. Para ello necesita invocar el servicio `deregisterProcess` del SF y el servicio *leaveRol* del OMS y habrá dejado el Mercado de Turismo. En este caso, no borra la descripción del servicio ya que no ha sido el quien la ha publicado, sino

que solo ha ofrecido ese servicio. Para ejecutar este escenario hay que pulsar en el botón `Deshacer` de la interfaz gráfica de `Funciones del Proveedor` realizandose las siguientes acciones:

1. Elimina la implementación del servicio que ofrece.
2. Deja el sistema, dejando de jugar el rol "`member`"dentro de la organización "`virtual`".

Y obteniendo el siguiente resultado por consola:

```
[ProviderAgent1] Vamos a salir...

[ProviderAgent1] Eliminarme como provedor del servicio SearchCheapTrain:
155@148-qpid://ProviderAgent1@localhost:8080

[ProviderAgent1] Salir de THOMAS: Ok
```

Deshacer Mercado de Turismo

En este escenario el agente `GodAgent` decide, debido a algún evento en el sistema (concretamente no hay ninguna agencia ni clientes en el sistema), deshacer el Mercado de Turismo. Primero elimina la descripción y la implementación de los servicios que ofrece `NewAgency`, `NewClient` y `NewProvider` del SF. A continuación, va comprobando organización por organización si hay algún agentes que aún desempeñar algún rol dentro de alguna agencia, y si no hay ninguno, elimina los roles que hay asociados a esa organización y luego la organización. Después elimina los roles que quedan en la organización `TourismMarket` como son `provider` y `customer`. Por último, deja el rol `MarketManager` de la organización `TourismMarket` y así puede eliminar la organización `TourismMarket`. Finalmente, dejará THOMAS y el sistema, habiéndose disuelto por completo el Mercado de Turismo. La lista de acciones que realiza el agente `GodAgent` son las siguientes:

1. Por cada organización O que haya la organización `TourismMarket`:
   - Por cada rol R que haya en la organización O.
     - Eliminar rol R de la organización O.
2. Deja el rol `MarketManager` de la organización `TourismMarket` y lo elimina.

3. Elimina los roles `provider` y `customer` de la organización `TourismMarket`.

4. Borra la organización `TourismMarket` del sistema.

5. Sale de THOMAS, dejando el rol "`member`" de la organización "`virtual`".

Para ello, hay que pulsar en el botón `Deshacer` de la interfaz gráfica de `Funciones de TourismMarket` obteniendose el siguiente resultado por consola.

```
[GodAgent] Eliminarme como provedor del servicio NewClient: 150

[GodAgent] Eliminar el profile del servicio NewClient : 0

[GodAgent] Eliminarme como provedor del servicio NewAgeny: 151

[GodAgent] Eliminar el profile del servicio NewAgency : 0

[GodAgent] Eliminarme como provedor del servicio NewProvider: 152

[GodAgent] Eliminar el profile del servicio NewProvider : 0

[GodAgent] Borrar rol AgencyManager de Agency2: Ok

[GodAgent] Borrar rol Provider de Agency2: Ok

[GodAgent] Borrar Agency2: Ok

[GodAgent] Borrar rol AgencyManager de Agency1: Ok

[GodAgent] Borrar rol Provider de Agency1: Ok

[GodAgent] Borrar Agency1: Ok

[GodAgent] Salir del rolo MarketManager de TourismMarket: Ok

[GodAgent] Borrar rol member de TourismMarket: Ok

[GodAgent] Borrar rol MarketManager de TourismMarket: Ok

[GodAgent] Borrar rol Payee de TourismMarket: Ok
```

```
[GodAgent] Borrar rol provider de TourismMarket: Ok

[GodAgent] Borrar rol customer de TourismMarket: Ok

[GodAgent] Borrar TourismMarket unit: Ok

[GodAgent] Salir de THOMAS: Ok
```

Capítulo 4

Conclusiones

En esta memoria se ha realizado un estudio de la integración de THOMAS y Magentix2 como plataforma especialmente orientada a soportar dinamicidad en Organizaciones Virtuales.

En el capítulo 2 hemos explicado qué son los Sistemas Multiagente, las Organizaciones Virtuales y las tecnologías THOMAS y Magentix2 las cuales hemos usado en nuestro proyecto. En el capítulo 3 hemos expuesto los requisitos que tendría que tener un Sistema Multiagente para que sea dinámico y hemos presentado un caso de estudio con el que hemos verificado la plataforma presentada anteriormente.

Al introducir la dinamicidad como propiedad en el sistema se ha permitido la evolución de los Sistemas Multiagente y facilitado su evolución en tiempo de ejecución. Esta arquitectura o plataforma representa un paso sólido para obtener verdaderamente Organizaciones Virtuales dinámicas. Además, sirve de base para comenzar a pensar en mecanismos que permitan adaptabilidad, es decir, sistemas que a partir de un evento sean capaces de adaptarse a ese cambio.

Con la ayuda del caso de estudio diseñado, implementado y testeado, hemos comprobado que THOMAS/Magentix2 ofrece dinamicidad y que es una clara opción, como plataforma, para desarrollar Sistemas Multiagente basados en Organizaciones Virtuales. Hemos verificado la especificación de THOMAS/Magentix2 y mejorado tanto las posibilidades como la API que ofrece la plataforma. A lo largo del trabajo hemos detectado y corregido diversos errores de la plataforma. Gracias a esta depuración disponemos de un producto más fiable y robusto.

Los resultados de este trabajo han sido satisfactoriamente publicados y expuestos en el congreso internacional *9th International Conference on Practical Applications of Agents and Multi-Agent Systems* [10]

Además, el trabajo realizado forma parte del trabajo que se ha desarrollado dentro del proyecto Consolider Ingenio (CSD2007-00022) *Agreement Technologies* cuyo responsable es Carlos Sierra y dentro del proyecto del Ministerio de Ciencia e Innovación (TIN2009-13839-C03-01) *Organizaciones Virtuales Adaptativas: Arquitecturas y Métodos de Desarrollo* cuyo responsable es Vicente J. Julian Inglada. También ha servido de apoyo para el proyecto *MAGENTIX II: Una plataforma para sistemas multiagente abiertos* del Ministerio de Ciencia e Innovación (TIN2008-04446/TIN) cuyo responsable es Ana María García Fornes. Por último ha servido para crear una prueba unitaria para la plataforma que será usada para comprobar si después de una nueva versión sigue ofreciendo las mismas funcionalidades con respecto al framework de THOMAS. Por último ha servido de guía y apoyo para el trabajo realizado con agentes móviles en Android por Jorge Luis Agüero Medina.

Como trabajo futuro consideramos la realización de la prueba de algún mecanismo de composición de servicios. Un agente detecta que hay una serie de servicios que son muy solicitados y crea un servicio compuesto por esos servicios. También podemos incluir aspectos de seguridad en la plataforma así como mejorar la interoperabilidad del caso de estudio con el usuario, es decir, mostrar de una forma gráfica todo la información y crear más y mejores servicios.

Apéndices

Apéndice A

Prueba unitaria de Magentix2 y THOMAS

Tourism Market: Ejemplo de desarrollo en Mag2THM y JADE

Alfonso A. Machado Benito
amachado@dsic.upv.es

Noviembre de 2010

Índice

1. Introducción

Es un ejemplo de desarrollo de organizaciones virtuales abiertas, hecho sobre Magentix 2, THOMAS y JADE, en el que se explotan las funcionalidades de Magentix 2 y de THOMAS. Sirve como prueba para comprobar que las funcionalidades de Mag2THM siguen funcionando. Su ejecución es un poco compleja, aunque se explica en un artículo que se publicará en poco tiempo. En este ejemplo se comprueban casi todos los métodos de THOMAS y se intenta exprimir al máximo la potencia de Magentix2.

1.1. Estructura de directorios

Dentro del proyecto `TestTourismMarket` encontraremos un conjunto de paquetes

- **src/agentesJ** Código de los agentes en JADE
  - **ClientAgentJADE.java** Agente en JADE que ejecuta el servicio dado en el p̈arametroq̈ue está en el método ClientAgentJADE
  - **ManagerAgencyJADE.java** Agente en JADE que crea una organización dentro de TourismMarket y ofrece un servicio nuevo.
- **src/agentesM** Código de los agentes Magentix2
  - **ClientAgent.java** Agente que ejecuta el servicio que se le pase en el parametro del constructor.
  - **GodAgent.java** Este agente es el que crea toda la infraestructura necesaria para la creación del Mercado de Turismo (Roles, unidades y servicios). Ademas es quien ofertara los servicios de NewClient, NewProvider y NewAgency.
  - **ManagerAgency1.java** Agente que creará una Agencia (Agency1) dentro de MarketManager y ofrecerá el servicio SearchCheapHotel, además tiene un evento que permite registrar un profile del servicio SearchCheapTrain, para que venga otro agente y lo ofrezca dentro de la agencia creada por él.
  - **ManagerAgency2.java** Agente que creará una Agencia (Agency2) dentro de MarketManager y ofrecerá el servicio SearchCheapFlight
  - **Provider1Agent.java** Agente que ofrece el servicio SearchCheapTrain. Necesita que este publicado el profile del servicio por Agency1Manager.
- **src/comportamientos** Comportamientos para los agentes de JADE. El nombre indica a que método del OMS o del SF atienden. El Responder.java corresponde con el comportamiento para atender las peticiones del servicio que registra el agente: ManagerAgencyJADE.java
- **src/extras**
  - **Agency1UI.java** Interfaz gráfica para el agente Agency1Manager.java que tiene el boton de deshacer la organización y de añadir el profile del servicio SearchCheapTrain
  - **GeneraAgentes.java** Una función que genere el texto que se necesita poner en el lanzador de JADE para que genere n agentes y poder hacer pruebas de carga.
  - **OMSAgentInform.java** Clase que usan los Agentes de JADE para guardar la información de la ubicación del OMS
  - **SFAgentInform.java** Clase que usan los Agentes de JADE para guardar la información de la ubicación del SF
  - **Salir.java** Interfaz gráfica para el resto de agentes de Magentix2 que tiene un botón que representa el concepto de que se produce un evento y la organización se deshace.
- **src/lanzadores**
  - **RunAgency1.java** Lanza el agente Agency1Manager.
  - **RunAgency2.java** Lanza el agente Agency2Manager.

- **RunClient.java** Lanza el agente ClientAgent.
- **RunMainOrganization.java** Lanza la plataforma Magentix2, limpia la base de datos, lanza los agentes pasarelas y el OMS y el SF. Ademas lanza el agente GodAgent y Agency1Manager.
- **RunMarketManager.java** Lanza el agente GodAgent.
- **RunProvider.java** Lanza el agente ProviderAgent.

- **configuration** Dentro de este directorio encontraremos dos ficheros de configuración.
  - loggin.xml: para poder configurar el log de las ejecuciones, aqui podremos decirle donde guardar el log.
  - Settings.xml: fichero de configuración donde podremos establecer los valores de conexión al broker (host, port ...).

1.2. Requisitos iniciales

- Si no están marcados los proyectos como Maven, botón derecho→Maven →Enable Dependency Management. Hay que asegurarse de que el proyecto está preparado para la versión de java 1.6 y no otra (ver en las propiedades del proyecto) porque a veces Maven cambia estas preferencias.
- MYSQL
- JADE
- Tomcat
- Servicios web de Thomas ejecutándose bajo Tomcat.
- Los servicios necesarios para el ejemplo `TourismMarket`, `SearchCheapHotel`, `SearchCheapFlight` y `SearchCheapTrain` ejecutándose bajo Tomcat. **Estos servicios fueron desarrollados concretamente para este ejemplo**, los cuales se pueden encontrar en Internet[1].
- Plataforma Magentix compilada y en el repositorio local de Maven. Para esto necesitamos hacer un `mvn install` en el proyecto *magentix2*. Si estamos en eclipse y tenemos el proyecto *magentix2* abierto, podemos hacer directamente botón derecho→Run As→Maven Install. Con esto se compilará el proyecto *magentix2*, se empaquetara en dos jar (uno solo con Magentix y otro con Magentix y todas sus dependencias) y se insertará el jar en el repositorio local.

Para los requisitos de MYSQL, Tomcat y los servicios web se puede ver la guía de pruebas unitarias de Magentix[2], donde se explica con detalle como configurar estos puntos. Si no tenemos disponible el paquete de Magentix, podemos encontrar los war correspondientes en subversion, bajo las siguientes subramas del repositorio https://gti-ia.dsic.upv.es/svn/thomas/

- sfservices/ Servicio SF de THOMAS.
- omsservices/ Servicio OMS de THOMAS.
- magentix2/ Plataforma Magentix2.
- TestTourismMarket/
  - TourismMarket/ Código del ejemplo a probar
  - WebServices/ Servicios necesarios para ejecutar el ejemplo

Bajo estos directorios están los fuentes para construir los respectivos war y el código de todo lo necesario. Para construirlos los war's, en el directorio del OMS o del SF ejecutaríamos `mvn package`, que dejaría el war correspondiente en un directorio `target` dentro del subdirectorio donde ejecutemos el comando. En el caso de sfservices, por ejemplo, tendríamos:

[1] https://gti-ia.dsic.upv.es/svn/thomas/TestTourismMarket/WebServices

[2] En la guía de Magentix se explican también como configurar correctamente Magentix para poder crear sin problemas el *jar* asociado e instalarlo en el repositorio local de Maven

```
usuario@maquina:~/workspace/sfservices/target$ ls
classes  sfservices  sfservices.war  war
```

- Si se desea tener una interfaz gráfica para visualizar la información del OMS y el SF será necesario tener y usar ThomasGUI [3].

2. Pruebas sobre Magentix2

Se necesita tener en ejecución el servidor de bases de datos MYSQL y Tomcat con los servicios antes mencionados. La prueba consistiría en hacer uso de las funciones de organizaciones de Thomas (unidades organizativas, roles, ...) desde agentes funcionando en Mag2THM o en JADE.

Dado que la prueba consiste en seguir cierto orden en la ejecución de comandos, no se puede automatizar. Los pasos iniciales para reproducir la prueba son los siguientes:

- Reiniciar el servidor Tomcat. Con esto nos aseguramos de que los servicios web están *limpios*. Este paso previo no se hace debido a este test, se hace debido al diseño de los servicios de Thomas. Es el mejor modo de asegurarnos que la salida será la misma a la esperada si todo va bien.
- Si usamos `ThomasGUI` como visualizador tendremos que hacer lo siguiente antes de ejecutar la prueba:
  - Ejecutar `ThomasGUI`. Debemos ver una ventana similar a la de la imagen 3.

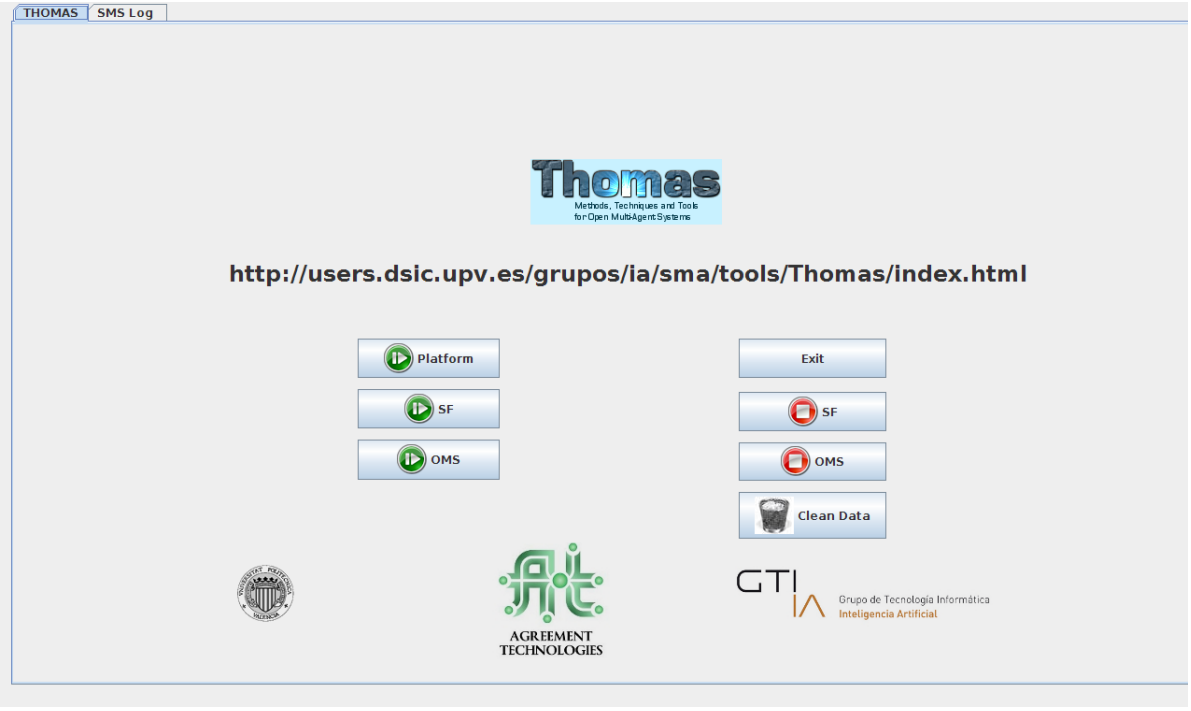

Figura 1: Captura de pantalla del arranque de `ThomasGUI`

En orden, apretar:

1. Clean Data
2. Platform
3. SF
4. OMS

[3] En la guía *Pruebas unitarias Demo Thomas (usada en el AAMAS'10)* se explican como configurarlo correctamente

Debemos ver en la consola de eclipse:

```
[Thread-5] core.BridgeAgentOutIn (BridgeAgentOutIn.java:71) - BridgeAgentOutIn waiting ...
[Thread-6] organization.SF (SF.java:602) - Agent SF active
[Thread-8] organization.OMS (OMS.java:794) - Agent OMS active
```

- Ejecutar el lanzador `RunMarketManager`.

```
[GodAgent]Entrar en THOMAS: Ok

[GodAgent]Register Unit TourismMarket result: Ok

[GodAgent]Register Role member result: Ok

[GodAgent]Register Role MarketManager result: Ok

[GodAgent]Acquire Role MarketManager result: Ok

[GodAgent]Register Role Payee result: Ok

[GodAgent]Register Role provider result: Ok

[GodAgent]Register Role customer result: Ok

[GodAgent] Register NewClientProfile result: 150

[GodAgent] Register NewClientProcess result: 150@144-qpid://GodAgent@localhost:8080

[GodAgent] Register NewAgencyProfile result: 151

[GodAgent] Register NewAgencyProcess result: 151@145-qpid://GodAgent@localhost:8080

[GodAgent] Register NewProviderProfile result: 152

[GodAgent] Register NewProviderProcess result: 152@146-qpid://GodAgent@localhost:8080
```

- Ejecutar el lanzador `RunAgency1`.

```
[ManagerAgency1] Entrar en THOMAS: Ok

[ManagerAgency1] Añado mi AID y el nombre de la agencia a crear como parametro:
qpid://ManagerAgency1@localhost:8080 , Agency1

[ManagerAgency1] Agente que ofrece el servicio NewAgency: GodAgent

[ManagerAgency1] Ejecutamos el servicio.

[ManagerAgency1] Valores devueltos.
      NewRol = AgencyManager
      ErrorValue = Ok!

[ManagerAgency1] Entrar en la Agencia Agency1 con el rol AgencyManager: Ok

[ManagerAgency1]Register SearchCheapHotel Profile result: 153

[ManagerAgency1] Register SearchCheapHotel Process result:
153@147-qpid://ManagerAgency1@localhost:8080
```

- Si no usamos `ThomasGUI` ejecutaremos el lanzador `RunMainOrganization`

```
[ManagerAgency1] Entrar en THOMAS: Ok

[GodAgent]Entrar en THOMAS: Ok

[GodAgent]Register Unit TourismMarket result: Ok

[GodAgent]Register Role member result: Ok

[GodAgent]Register Role MarketManager result: Ok

[GodAgent]Acquire Role MarketManager result: Ok

[GodAgent]Register Role Payee result: Ok

[GodAgent]Register Role provider result: Ok

[GodAgent]Register Role customer result: Ok

[GodAgent] Register NewClientProfile result: 150

[GodAgent] Register NewClientProcess result:
150@144-qpid://GodAgent@localhost:8080

[GodAgent] Register NewAgencyProfile result: 151

[GodAgent] Register NewAgencyProcess result:
151@145-qpid://GodAgent@localhost:8080

[ManagerAgency1] Añado mi AID y el nombre de la agencia a crear como parametro:
qpid://ManagerAgency1@localhost:8080 , Agency1

[ManagerAgency1] Agente que ofrece el servicio NewAgency: GodAgent

[ManagerAgency1] Ejecutamos el servicio.

[GodAgent] He recibido una petición de mi servicio NewAgency
[GodAgent] AGREE

[GodAgent] Executing... [Agency1, qpid://ManagerAgency1@localhost:8080]

[GodAgent] NewAgency: Registro de la nueva unidad, Agency1: Ok
[GodAgent] NewAgency: Registro del nuevo rol AgencyManager: Ok
[GodAgent] NewAgency: adquiere el rol el agente creador de la Agencia:
{http://localhost:8080/TourismMarket/owl/owls/NewAgencyProcess.owl#NameUnit=Agency1,
http://localhost:8080/TourismMarket/owl/owls/NewAgencyProcess.owl
#AgentID=qpid://ManagerAgency1@localhost:8080}
[GodAgent] NewAgency: Registro del nuevo rol Provider: Ok
[GodAgent]Creating Inform Message...
[ManagerAgency1] Valores devueltos.
      NewRol = AgencyManager
      ErrorValue = Ok!

[ManagerAgency1] Entrar en la Agencia Agency1 con el rol AgencyManager: Ok
```

```
[GodAgent] Register NewProviderProfile result: 152

[GodAgent] Register NewProviderProcess result: 152@146-qpid://GodAgent@localhost:8080

[ManagerAgency1]Register SearchCheapHotel Profile result: 153

[ManagerAgency1] Register SearchCheapHotel Process result:
153@147-qpid://ManagerAgency1@localhost:8080
```

Al acabar de ejecutarse una cosa o la otra conceptualmente tendremos lo que muestra la figura 2

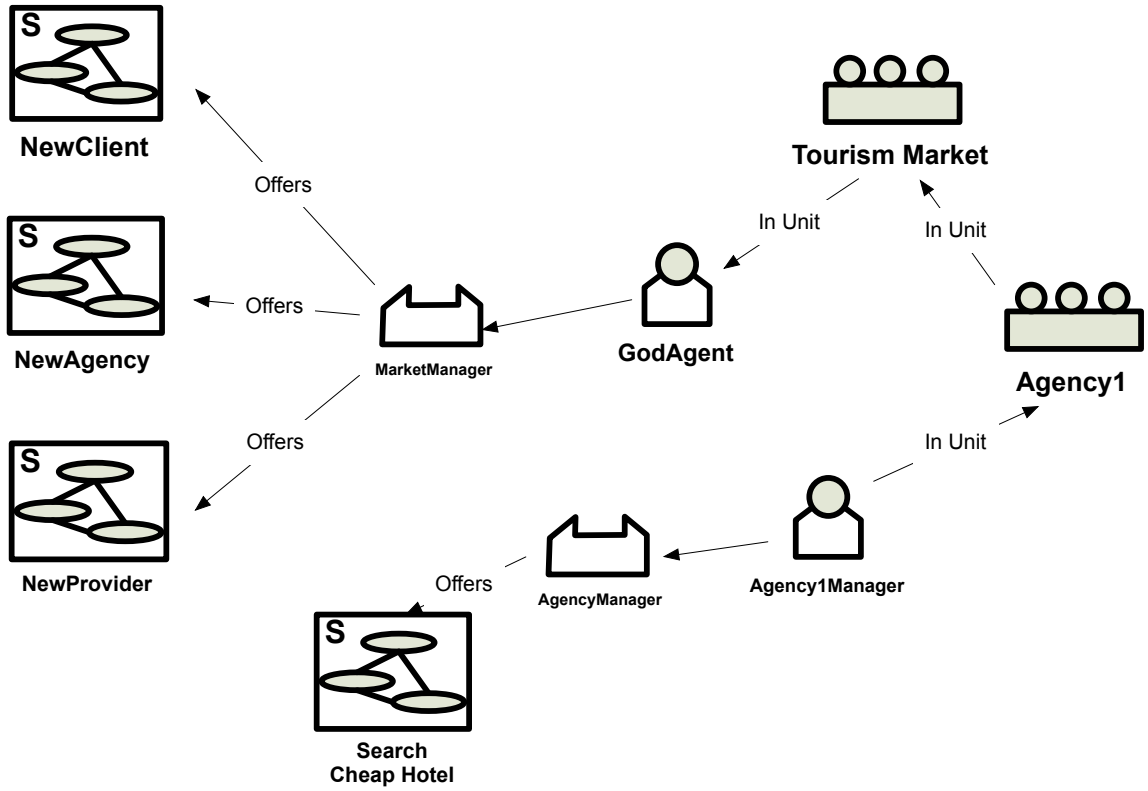

Figura 2: Estructura principal del Mercado de Turismo

2.1. Añadir una segunda Agencia

Probaremos la creación de una segunda agencia dentro de la organización desde Mag2THM. Para ellos introducimos en la plataforma el agente `Agency2Manager` que creará la agencia `Agency2` ofreciendo el servicio `SearchCheapFlight`

- Ejecutar el lanzador `RunAgency2`

```
[ManagerAgency2] Entrar en THOMAS: Ok

[ManagerAgency2] Añado mi AID y el nombre de la agencia a crear como parametro:
qpid://ManagerAgency2@localhost:8080 , Agency2

[ManagerAgency2] Agente que ofrece el servicio NewAgency: GodAgent

[ManagerAgency2] Ejecutamos el servicio.

[ManagerAgency2] Valores devueltos.
      NewRol = AgencyManager
      ErrorValue = Ok!

[ManagerAgency2] Entrar en la Agencia Agency2 con el rol AgencyManager: Ok

[ManagerAgency2]Register SearchCheapFlight Profile result: 154

[ManagerAgency2] Register SearchCheapFlgiht Process result:
154@147-qpid://ManagerAgency2@localhost:8080
```

2.2. Ejecutar el Proveedor

Probaremos lanzar un agente que buscara si alguien quiere el servicio que el puede ofrecer, si es así se unirá a esa agencia y lo ofrecerá. Para ello primero tendremos que hacer click en añadir profile de la interfaz gráfica de Agency1 como la que muestra la figura 3.

Figura 3: Interfaz gráfica del agente `ManagerAgency1`

Al hacer click en la consola aparecerá:

```
[ManagerAgency1] Register SCT  Profile result: 155
```

Ahora tenemos que introducir el agente `ProviderAgent`.

- Ejecutar el lanzador `RunProvider`

```
[ProviderAgent1] Entrar en THOMAS: Ok

[ProviderAgent1] Anyado mi AID como parametro: qpid://ProviderAgent1@localhost:8080

[ProviderAgent1] Agente que ofrece el servicio NewProvider: GodAgent

[ProviderAgent1] Ejecutamos el servicio.

[ProviderAgent1] Valores devueltos.
      NewUnit = TourismMarket
      NewRol = provider
      ErrorValue = Ok!
```

```
[ProviderAgent1] Entrar en TourismMarket: Ok

[ProviderAgent1] Comprobamos si alguien quiere el servicio SearchCheapTrain

[ProviderAgent1] Alguien quiere el servicio SearchCheapTrain y esta registrado en el SF con id: 155

[ProviderAgent1] Adquirir rol customer: Ok

[ProviderAgent1] Register SearchCheapTrain Process result: 155@148-qpid://ProviderAgent1@localhost:8080
```

2.3. Ejecutar un cliente

En este punto de la ejecución tendremos la organización *TourismMarket* como muestra la figura **??** (espero que sirva para tener una visión más clara de lo que se ha conseguido). Ahora probaremos lanzar un cliente en Mag2THM que ejecute los tres servicio que hay ofrecidos. Para ello introducimos en la plataforma el agente `ClientAgent` que ejecutará el servicio que le digamos en el parámetro de creación del agente.

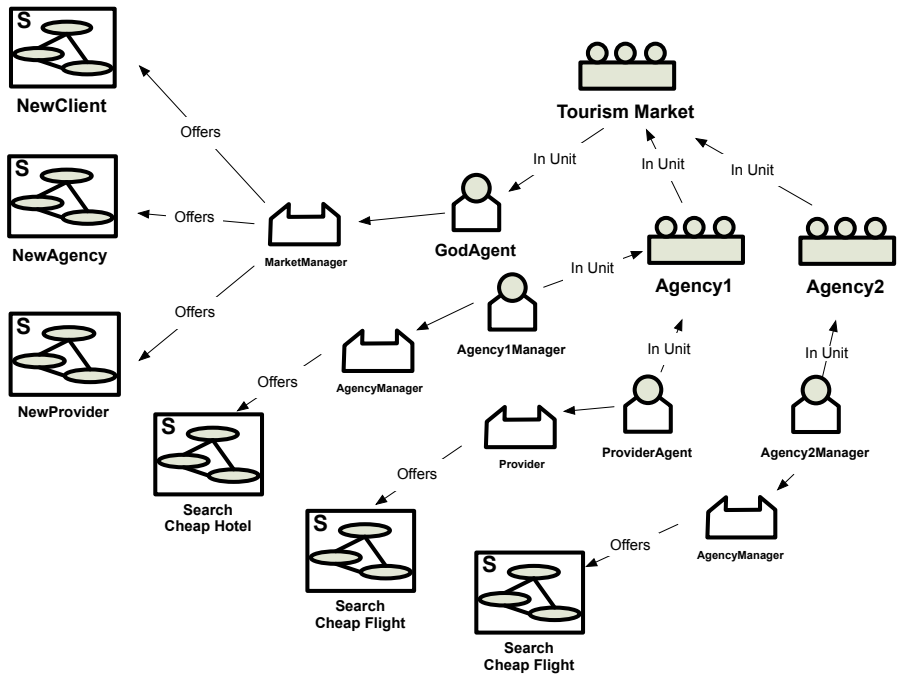

Figura 4: Estructura de TourismMarket

- Ejecutar el lanzador `RunClient`

```
[ClientAgent1] Entrar en THOMAS: Ok

[ClientAgent2] Entrar en THOMAS: Ok

[ClientAgent3] Entrar en THOMAS: Ok

[ClientAgent1] Añado mi AID como parametro: qpid://ClientAgent1@localhost:8080
```

```
[ClientAgent1] Agente que ofrece el servicio NewClient: GodAgent

[ClientAgent1] Ejecutamos el servicio.

[ClientAgent1] Valores devueltos.
      NewUnit = TourismMarket
      NewRol = customer
      ErrorValue = Ok!
[ClientAgent1] Entrar en TourismMarket: Ok

[ClientAgent2] Añado mi AID como parametro: qpid://ClientAgent2@localhost:8080

[ClientAgent2] Agente que ofrece el servicio NewClient: GodAgent

[ClientAgent2] Ejecutamos el servicio.

[ClientAgent2] Valores devueltos.
      NewUnit = TourismMarket
      NewRol = customer
      ErrorValue = Ok!

[ClientAgent2] Entrar en TourismMarket: Ok

[ClientAgent3] Añado mi AID como parametro: qpid://ClientAgent3@localhost:8080

[ClientAgent3] Agente que ofrece el servicio NewClient: GodAgent

[ClientAgent3] Ejecutamos el servicio.

[ClientAgent3] Valores devueltos.
      NewUnit = TourismMarket
      NewRol = customer
      ErrorValue = Ok!

[ClientAgent3] Entrar en TourismMarket: Ok

[ClientAgent1] Añadimos los parametros de entrada:...

[ClientAgent1] Agente que ofrece el servicio SearchCheapFlight: ManagerAgency2

[ClientAgent1] Valores devueltos.
      SearchCheapFlightOutputFlightCompany = Oceanic
      SearchCheapFlightOutputFlight = 815
      SearchCheapFlightOutputPrice = 750

[ClientAgent1] Vamos a salir...

[ClientAgent2] Añadimos los parametros de entrada:...

[ClientAgent2] Agente que ofrece el servicio SearchCheapTrain: ProviderAgent1

[ClientAgent1] Salir de THOMAS : Ok

[ClientAgent2] Valores devueltos.
      SearchCheapTrainOutputTrain = Alaris
      SearchCheapTrainOutputDeparture = 10:15
      SearchCheapTrainOutputArrival = 13:15
      SearchCheapTrainOutputDuration = 3:00
      SearchCheapTrainOutputPrice = 47
```

```
[ClientAgent2] Vamos a salir...

[ClientAgent2] Salir de THOMAS : Ok

[ClientAgent3] Añadimos los parametros de entrada: Categoria = 2 , Pais = Spain y Ciudad = Valencia

[ClientAgent3] Agente que ofrece el servicio SearchCheapHotel: ManagerAgency1

[ClientAgent3] Valores devueltos.
      SearchCheapHotelOutputHotelCompany = NH
      SearchCheapHotelOutputHotel = Hotel Puerta de Valencia

[ClientAgent3] Vamos a salir...

[ClientAgent3] Salir de THOMAS : Ok
```

2.4. Deshacer toda la organización TourismMarket

Probaremos todos los métodos para borrar roles, unidades y servicios. Hay que seguir el orden que explico a continuación.

1. Salir de la organización Agency2 haciendo click en la interfaz gráfica de `Funciones de Agency2` en el botón `Deshacer`.

```
[ManagerAgency2] Eliminarme como provedor del servicio SearchCheapFlight:
154@147-qpid://ManagerAgency2@localhost:8080

[ManagerAgency2] Eliminar el profile del servicio SearchCheapFlight: 1

[ManagerAgency2] Salir de THOMAS: Ok
```

2. Que el Agente `ProviderAgent` deje de ofrecer el servicio y deje la organización. Para ello hacer click en el botón `Deshacer` de la interfaz gráfica de `Funciones del Proveedor`.

```
[ProviderAgent1] Vamos a salir...

[ProviderAgent1] Eliminarme como provedor del servicio SearchCheapTrain:
155@148-qpid://ProviderAgent1@localhost:8080

[ProviderAgent1] Dejar el rol customer de la organizacion tourismmarket: Ok
```

3. Deshacer la Agency1. Para ello hacer click en el botón `Deshacer` de la interfaz gráfica de `Funciones de Agency1`.

```
[[ManagerAgency1] Eliminarme como provedor del servicio SearchCheapHotel: 153

[ManagerAgency1] Eliminar el profile del servicio SearchCheapHotel: 1

[ManagerAgency1] Eliminar el profile del servicio nuevo: 0

[ManagerAgency1] Salir de THOMAS: Ok
```

4. Deshacer MarketManager. Para ello hacer click en el botón `Deshacer` de la interfaz gráfica de `Funciones de TourismMarket`.

```
[GodAgent] Eliminarme como provedor del servicio NewClient: 150

[GodAgent] Eliminar el profile del servicio NewClient : 0

[GodAgent] Eliminarme como provedor del servicio NewAgeny: 151

[GodAgent] Eliminar el profile del servicio NewAgency : 0

[GodAgent] Eliminarme como provedor del servicio NewProvider: 152

[GodAgent] Eliminar el profile del servicio NewProvider : 0

[GodAgent] Borrar rol AgencyManager de Agency2: Ok

[GodAgent] Borrar rol Provider de Agency2: Ok

[GodAgent] Borrar Agency2: Ok

[GodAgent] Borrar rol AgencyManager de Agency1: Ok

[GodAgent] Borrar rol Provider de Agency1: Ok

[GodAgent] Borrar Agency1: Ok

[GodAgent] Salir del rolo MarketManager de TourismMarket: Ok

[GodAgent] Borrar rol member de TourismMarket: Ok

[GodAgent] Borrar rol MarketManager de TourismMarket: Ok

[GodAgent] Borrar rol Payee de TourismMarket: Ok

[GodAgent] Borrar rol provider de TourismMarket: Ok

[GodAgent] Borrar rol customer de TourismMarket: Ok

[GodAgent] Borrar TourismMarket unit: Ok

[GodAgent] Salir de THOMAS: Ok
```

3. Pruebas con Magentix2 y JADE

3.1. Como ejecutar un agente en JADE

Primero creamos un lanzador de JADE. Hacemos click en **Run configuartion** como se ve en la figura 5, nos aparecera una ventana como la que muestra la figura 6 , y rellenaremos los campos *Project* con el nombre del proyecto donde esté el código de lo agentes JADE y *Main Class* con *jade.Boot*. Después haremos click en la pestaña *Arguments* y rellenaremos el campo *Program Arguments* como en la figura 7. Por último, si hacemos click en *Apply* y en *Run* habremos lanzado JADE y nos aparecerá la interfaz de JADE como muestra la figura 8.

Con JADE lanzado, tenemos que hacer click en *New Agent*, que es el icono que aparece en la IGU de JADE como el de la figura 9, estando seleccionado el contenedor principal como muestra la figura 8. Nos aparecera una ventana como la de la figura 10 donde pondremos el nombre del agente en el campo *Agent Name* y la ubicación de la clase del agente el campo *Class Name*, para saber la ruta de la clase nos podemos ayudar haciendo click en el botón con *"..."* que nos mostrará una lista de las clases que pertenecen a agentes que hay dentro del proyecto que hemos seleccionado anteriormente. Finalmente tendrá que quedar algo como lo que muestra la figura 11. Por último al hacer click en *Ok* el agente se lanzará y empezará a ejecutarse.

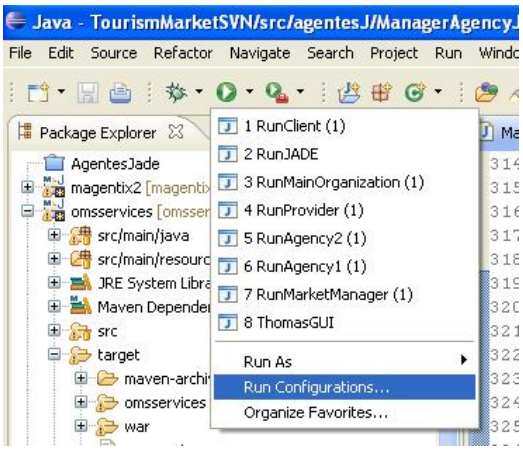

Figura 5: Crear un lanzador de JADE

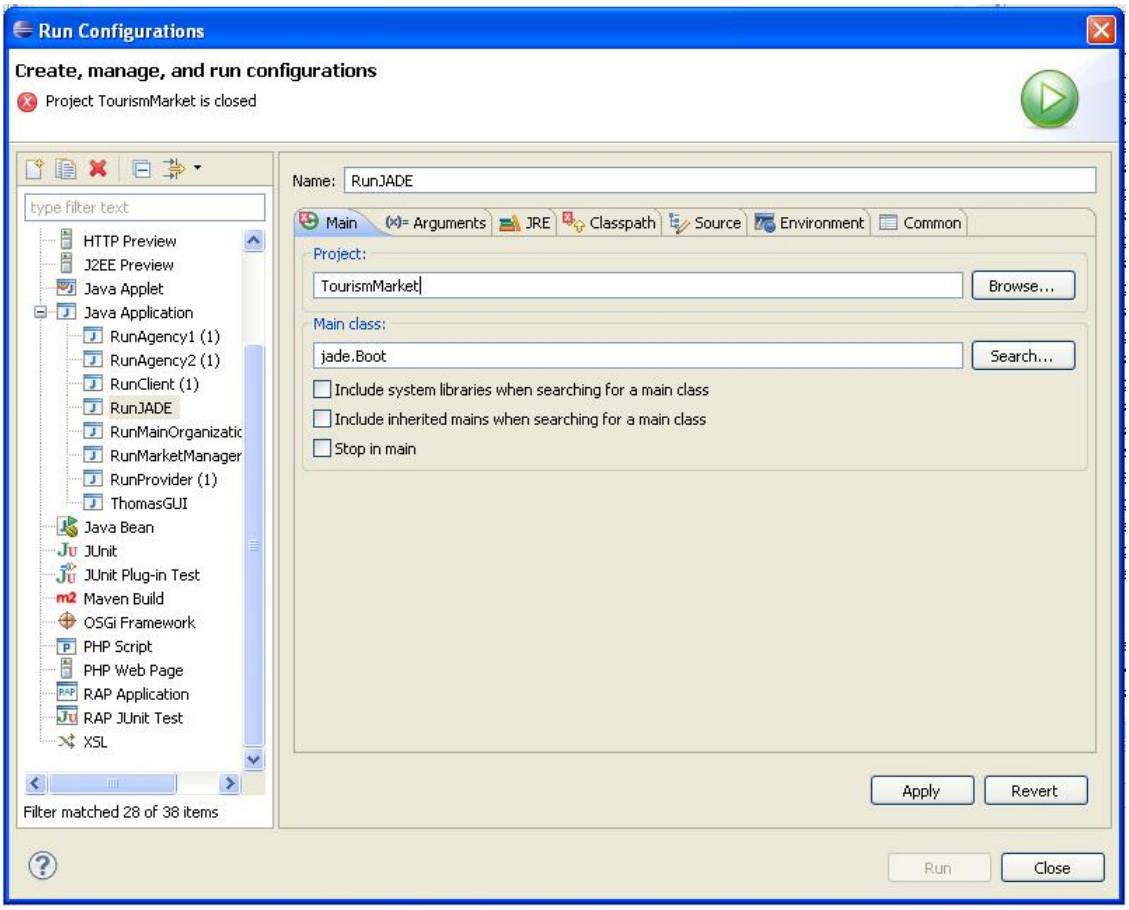

Figura 6: Crear un lanzador de JADE

3.2. Pruebas

Probaremos la creación de una segunda agencia desde JADE y un cliente en JADE que ejecutará un servicio ofrecido por un agente en Magentix2 y otro en JADE. también probaremos como un cliente en Magentix2 ejecuta un servicio ofrecido por un agente JADE. Lo primero que hay que hacer es lanzar la organización principal del Mercado de Turismo, como se explica en la sección 2 (Sin ejecutar la nueva agencia, ni el provedor, ni el cliente).

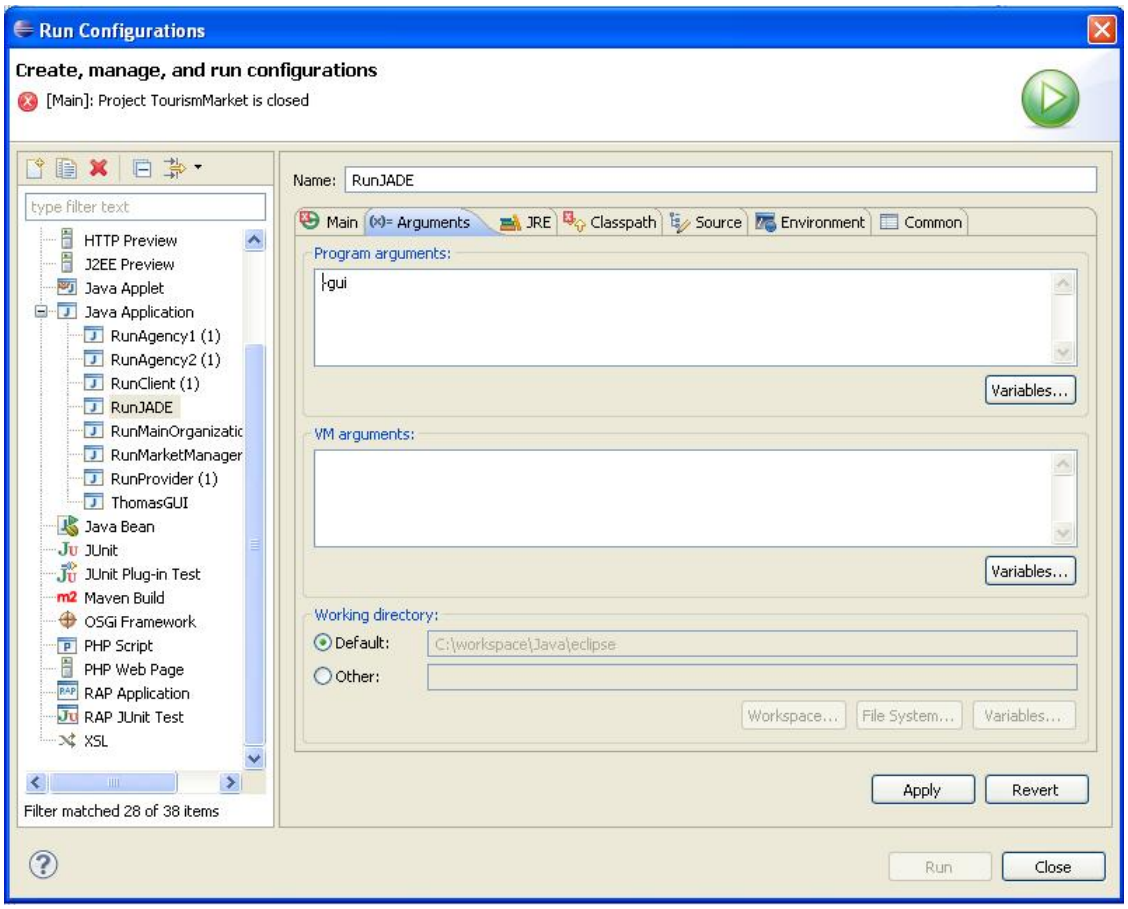

Figura 7: Crear un lanzador de JADE

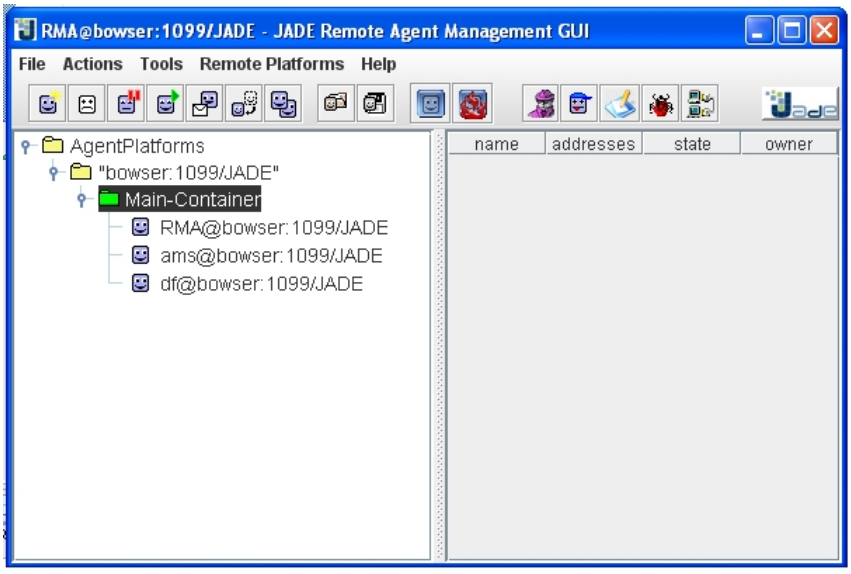

Figura 8: Interfaz de JADE

3.2.1. Crear una Agencia

Introducimos el agente `AgencyManagerJADE`[4] como se ha explicado en el sección 3.1. Este agente creará una agencia llamada *Agency2* y ofrecerá el servicio *SearchCheapFlight*.

[4]Este agente no deregistra el servicio ni es capaz de salir del sistema, es decir, crear la organización, registra el servicio que va a ofrecer y se queda indefinidamente ofreciendo el servicio en el sistema. En próximas mejoras se añadirá está funcionalidad

Figura 9: Botón de nuevo agente

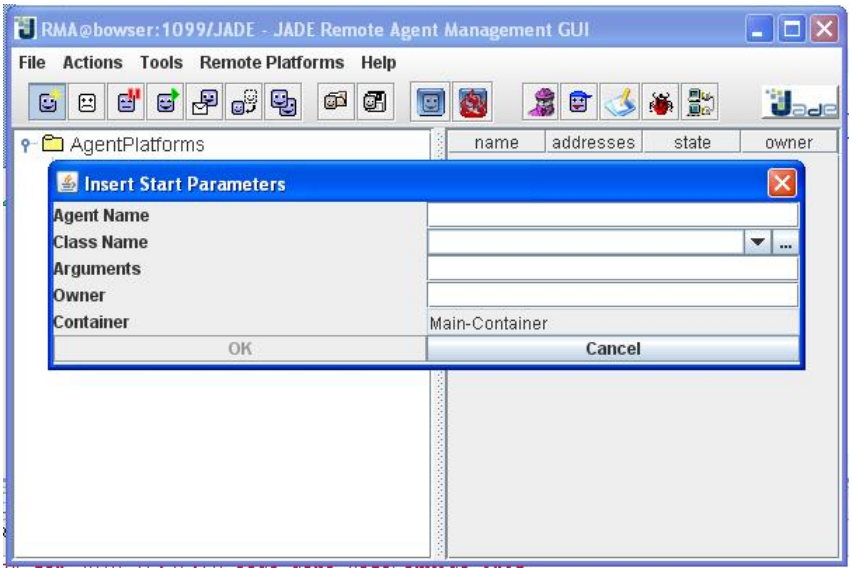

Figura 10: Parametros del nuevo agente

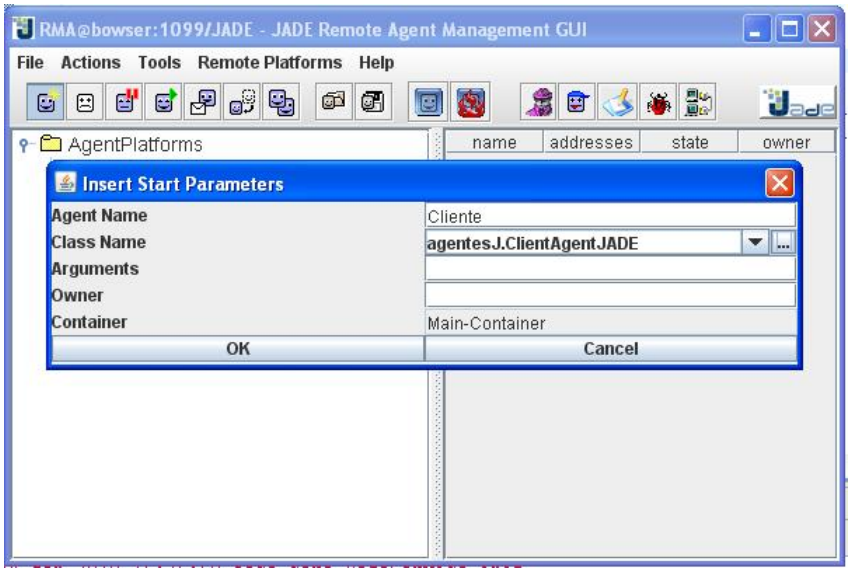

Figura 11: Lanzar nuevo agente en JADE

```
I'm the ManagerAgency@bowser:1099/JADE Agent!!!!
ManagerAgency: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/AcquireRoleProcess.owl
 RoleID=member UnitID=virtual AgentID=ManagerAgency@bowser:1099/JADE"
 :protocol  fipa-request
)
ManagerAgency: SearchService =(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/SearchServiceProcess.owl
 SearchServiceInputServicePurpose=NewAgency"
 :protocol  fipa-request
)
ManagerAgency: OOH! OMS Has agreed to excute the service AcquireRol!
ManagerAgency: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
ManagerAgency: OOH! SF Has agreed to excute the service SearchService!
ManagerAgency: GetProfile =(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/GetProfileProcess.owl
 GetProfileInputServiceID=12"
 :protocol  fipa-request
)
ManagerAgency: SF SearchService IDProfile = 12
ManagerAgency: OOH! SF Has agreed to excute the service GerProfile!
ManagerAgency: SF GetProfile URL = http://localhost:8080/TourismMarket/owl/owls/NewAgencyProfile.owl
ManagerAgency: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/AcquireRoleProcess.owl
 RoleID=member UnitID=tourismmarket AgentID=ManagerAgency@bowser:1099/JADE"
 :protocol  fipa-request
)
ManagerAgency: GetProcess =(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
 GetProcessInputServiceID=12"
 :protocol  fipa-request
)
ManagerAgency: OOH! OMS Has agreed to excute the service AcquireRol!
ManagerAgency: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
ManagerAgency: OOH! SF Has agreed to excute the service GetProcess!
ManagerAgency:SF has informed me of the status of my request. They said :
GetProcessProcess={http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
#GetProcessOutputProcessList=12@12-GodAgent
http://localhost:8080/TourismMarket/owl/owls/NewAgencyProcess.owl,
http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
#GetProcessOutputServiceStatus=1}
```

```
ManagerAgency: SF GetProcess Provider = GodAgent
[ManagerAgency]Sms to send: (REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name GodAgent@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/TourismMarket/owl/owls/NewAgencyProcess.owl
 AgentID=ManagerAgency@bowser:1099/JADE NameUnit=Agency2"
 :protocol  fipa-request
)
[ManagerAgency]Sending...
ManagerAgency: OOH! GodAgent Has agreed to excute the service GenericService!
ManagerAgency:GodAgent has informed me of the status of my request. They said :
NewAgencyProcess={http://localhost:8080/TourismMarket/owl/owls/NewAgencyProcess.owl
#NewRol=AgencyManager, http://localhost:8080/TourismMarket/owl/owls/NewAgencyProcess.owl
#ErrorValue=Ok!}
ManagerAgency: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/AcquireRoleProcess.owl
 RoleID=AgencyManager UnitID=Agency2 AgentID=ManagerAgency@bowser:1099/JADE"
 :protocol  fipa-request
)
ManagerAgency: SearchService =(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/SearchServiceProcess.owl
 SearchServiceInputServicePurpose=SearchCheapFlight"
 :protocol  fipa-request
)
ManagerAgency: OOH! OMS Has agreed to excute the service AcquireRol!
ManagerAgency: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
ManagerAgency: OOH! SF Has agreed to excute the service SearchService!
(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/RegisterProfileProcess.owl
 RegisterProfileInputServiceGoal= RegisterProfileInputServiceProfile=
 http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProfile.owl#SearchCheapFlight"
 :protocol  fipa-request
)
ManagerAgency: SF SearchService IDProfile =
ManagerAgency: OOH! SF Has agreed to excute the RegisterProfile!
ManagerAgency:SF(INFORM
 :sender  ( agent-identifier :name SF@bowser:8081
 :addresses (sequence http://bowser.dsic.upv.es:8081 ))
 :receiver  (set ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778 )) )
 :content  "RegisterProfileProcess={http://localhost:8080/sfservices/SFservices/owl/owls/
 RegisterProfileProcess.owl#RegisterProfileOutputServiceStatus=1,
 http://localhost:8080/sfservices/SFservices/owl/owls/
 RegisterProfileProcess.owl#RegisterProfileOutputServiceID=15}"
 :protocol  fipa-request
 :conversation-id  C29291718_1290456334963 )
```

```
(REQUEST
 :sender  ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/RegisterProcessProcess.owl
 RegisterProcessInputServiceModel=http://localhost:8080/SearchCheapFlight/owl/owls/
 SearchCheapFlightProcess.owl#SearchCheapFlightProcess RegisterProcessInputServiceID=15"
 :protocol  fipa-request
)
ManagerAgency:SF RegisterProfile has returned URL = 15
ManagerAgency: OOH! SF Has agreed to excute the RegisterProcess!
ManagerAgency:SF(INFORM
 :sender  ( agent-identifier :name SF@bowser:8081
 :addresses (sequence http://bowser.dsic.upv.es:8081 ))
 :receiver  (set ( agent-identifier :name ManagerAgency@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778 )) )
 :content  "RegisterProcessProcess={http://localhost:8080/sfservices/SFservices/owl/owls/
 RegisterProcessProcess.owl#RegisterProcessOutputServiceModelID=
 15@15-ManagerAgency@bowser:1099/JADE,
 http://localhost:8080/sfservices/SFservices/owl/owls/RegisterProcessProcess.owl
 #RegisterProcessOutputServiceStatus=1}"
 :protocol  fipa-request
 :conversation-id  C27253500_1290456338713 )
ManagerAgency:SF RegisterProcess has returned = 15@15-ManagerAgency@bowser:1099/JADE
```

3.2.2. Ejecutar un cliente en Magentix2

Ejecutaremos el lanzador *RunClient* modificado[5], para que solo ejecute el servicio *SearchCheapFlight*, como se ha explicado en la sección 2.3.

```
[ClientAgent1] Entrar en THOMAS: Ok

[ClientAgent2] Entrar en THOMAS: Ok

[ClientAgent2] Anyado mi AID como parametro: qpid://ClientAgent2@localhost:8080

[ClientAgent2] Agente que ofrece el servicio NewClient: GodAgent

[ClientAgent2] Ejecutamos el servicio.

[ClientAgent2] Valores devueltos.
      NewUnit = TourismMarket
      NewRol = customer
      ErrorValue = Ok!
[ClientAgent2] Entrar en TourismMarket: Ok

[ClientAgent1] Anyado mi AID como parametro: qpid://ClientAgent1@localhost:8080

[ClientAgent1] Agente que ofrece el servicio NewClient: GodAgent

[ClientAgent1] Ejecutamos el servicio.
```

[5]La modificación consiste en comentar la línea de la instrucción *servicios.add("SearchCheapTrain")* y cambiar la inicialización de la variable N por 2

```
[ClientAgent1] Valores devueltos.
      NewUnit = TourismMarket
      NewRol = customer
      ErrorValue = Ok!
[ClientAgent1] Entrar en TourismMarket: Ok

[ClientAgent2] Anyadimos los parametros de entrada: Categoria = 2 , Pais = Spain y Ciudad = Valencia

[ClientAgent2] Agente que ofrece el servicio SearchCheapHotel: ManagerAgency1

[ClientAgent2] Valores devueltos.
      SearchCheapHotelOutputHotelCompany = NH
      SearchCheapHotelOutputHotel = Hotel Puerta de Valencia

[ClientAgent2] Vamos a salir...

[ClientAgent2] Salir de THOMAS: Ok

[ClientAgent1] Anyadimos los parametros de entrada:...

[ClientAgent1] Agente que ofrece el servicio SearchCheapFlight: AgencyManagerJADE@bowser:1099/JADE

[ClientAgent1] Valores devueltos.
      SearchCheapFlightOutputFlightCompany = Oceanic
      SearchCheapFlightOutputFlight = 815
      SearchCheapFlightOutputPrice = 750?

[ClientAgent1] Vamos a salir...

[ClientAgent1] Salir de THOMAS: Ok
```

3.2.3. Ejecutar un cliente en JADE

Introducimos el agente `ClientAgentJADE` como se ha explicado en el sección 3.1. Este agente ejecutará el servicio que le digas en el *"parametro"* que tiene en el constructor (*ClientAgentJADE*).

- Ejecutar el servicio *SearchCheapFlight* para ver el funcionamiento de la ejecución de un agente JADE ejecutando un servicio en JADE

```
I'm the Client@bowser:1099/JADE Agent!!!!
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/AcquireRoleProcess.owl
 RoleID=member UnitID=virtual AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: SearchService =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/SearchServiceProcess.owl
 SearchServiceInputServicePurpose=NewClient"
 :protocol  fipa-request
)
```

```
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service SearchService!
Client: GetProfile =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/GetProfileProcess.owl
 GetProfileInputServiceID=11"
 :protocol  fipa-request
)
Client: SF SearchService IDProfile = 11
Client: OOH! SF Has agreed to excute the service GerProfile!
Client: SF GetProfile URL = http://localhost:8080/TourismMarket/owl/owls/NewClientProfile.owl

Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/AcquireRoleProcess.owl
 RoleID=member UnitID=tourismmarket AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: GetProcess =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
 GetProcessInputServiceID=11"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service GetProcess!
Client:SF has informed me of the status of my request. They said :
GetProcessProcess={http://localhost:8080/sfservices/SFservices/owl/owls/
GetProcessProcess.owl#GetProcessOutputProcessList=11@11-GodAgent
http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl,
http://localhost:8080/sfservices/SFservices/owl/owls/
GetProcessProcess.owl#GetProcessOutputServiceStatus=1}
Client: SF GetProcess Provider = GodAgent
Client: Ejecutando servicio...
[Client]Sms to send: (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name GodAgent@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
[Client]Sending...
Client: OOH! GodAgent Has agreed to excute the service GenericService!
Client:GodAgent has informed me of the status of my request. They said :
NewClientProcess={http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl
#NewRol=customer, http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl
#NewUnit=TourismMarket, http://localhost:8080/TourismMarket/owl/owls/
NewClientProcess.owl#ErrorValue=Ok!}
```

```
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/AcquireRoleProcess.owl
 RoleID=customer UnitID=TourismMarket AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: SearchService =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/SearchServiceProcess.owl
 SearchServiceInputServicePurpose=SearchCheapFlight"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service SearchService!
Client: GetProfile =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/GetProfileProcess.owl
 GetProfileInputServiceID=15"
 :protocol  fipa-request
)
Client: SF SearchService IDProfile = 15
Client: OOH! SF Has agreed to excute the service GerProfile!
Client: SF GetProfile URL = http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProfile.owl

Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/AcquireRoleProcess.owl
 RoleID=customer UnitID=tourismmarket AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: GetProcess =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
 GetProcessInputServiceID=15"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Error ErrorValue = Duplicate
Client: OOH! SF Has agreed to excute the service GetProcess!
Client:SF has informed me of the status of my request. They said :
GetProcessProcess={http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
```

```
#GetProcessOutputProcessList=15@15-ManagerAgency@bowser:1099/JADE
http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl,
http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
#GetProcessOutputServiceStatus=1}
Client: SF GetProcess Provider = ManagerAgency@bowser:1099/JADE
Client: Ejecutando servicio...
[Client]Sms to send: (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name ManagerAgency@bowser:1099/JADE ) )
 :content  "http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl
 SearchCheapFlightInputDepartingFrom=Sydney
 SearchCheapFlightInputGoingTo=Nueva York
 SearchCheapFlightInputDepartDate=22/10/00
 SearchCheapFlightInputReturnDate=10/11/00
 SearchCheapFlightInputNumberOfPassanger=2"
 :protocol  fipa-request
)
[Client]Sending...
ManagerAgency: Doc OWL-S: http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl
ManagerAgency: AGREE
ManagerAgency: Sending First message:(AGREE
 :receiver  (set ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc )) )
 :content  "SearchCheapFlightProcess=Agree"
 :reply-with  Client@bowser:1099/JADE1290456368635  :protocol  fipa-request
 :conversation-id  C1142607_1290456368635 )
Client: OOH! ManagerAgency Has agreed to excute the service GenericService!
ManagerAgency: Doc OWL-S: http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl
ManagerAgency: Executing... [Sydney, 10/11/00, 22/10/00, 2, Nueva]
ManagerAgency: Values obtained... :
http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl
#SearchCheapFlightOutputPrice=750,
http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl
#SearchCheapFlightOutputFlightCompany=Oceanic,
http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl
#SearchCheapFlightOutputFlight=815
ManagerAgency: Creating inform message to send...
ManagerAgency: Before set message content...
Client:ManagerAgency has informed me of the status of my request. They said :
SearchCheapFlightProcess=http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl
#SearchCheapFlightOutputPrice=750,
http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl
#SearchCheapFlightOutputFlightCompany=Oceanic,
http://localhost:8080/SearchCheapFlight/owl/owls/SearchCheapFlightProcess.owl
#SearchCheapFlightOutputFlight=815
[Client] Valores devueltos.
      SearchCheapFlightOutputFlightCompany = Oceanic
      SearchCheapFlightOutputFlight = 815
      SearchCheapFlightOutputPrice = 750
Client: Salir de THOMAS
Client: LeaveRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/LeaveRoleProcess.owl
 RoleID=member UnitID=virtual AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
```

```
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
```

- Ejecutar el servicio *SearchCheapHotel* para ver el funcionamiento de la ejecución de un agente JADE ejecutando un servicio en Magentix2

```
I'm the Client@bowser:1099/JADE Agent!!!!
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/AcquireRoleProcess.owl
 RoleID=member UnitID=virtual AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: SearchService =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/SearchServiceProcess.owl
 SearchServiceInputServicePurpose=NewClient"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service SearchService!
Client: GetProfile =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/GetProfileProcess.owl
 GetProfileInputServiceID=26"
 :protocol  fipa-request
)
Client: SF SearchService IDProfile = 26
Client: OOH! SF Has agreed to excute the service GerProfile!
Client: SF GetProfile URL = http://localhost:8080/TourismMarket/owl/owls/NewClientProfile.owl

Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/AcquireRoleProcess.owl
 RoleID=member UnitID=tourismmarket AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
```

```
Client: GetProcess =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
 GetProcessInputServiceID=26"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service GetProcess!
Client:SF has informed me of the status of my request. They said :
GetProcessProcess=http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
#GetProcessOutputProcessList=26@25-GodAgent
http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl,
http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
#GetProcessOutputServiceStatus=1
Client: SF GetProcess Provider = GodAgent
Client: Ejecutando servicio...
[Client]Sms to send: (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name GodAgent@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl
 AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
[Client]Sending...
Client: OOH! GodAgent Has agreed to excute the service GenericService!
Client:GodAgent has informed me of the status of my request. They said :
NewClientProcess=http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl
#NewRol=customer,
http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl
#NewUnit=TourismMarket,
http://localhost:8080/TourismMarket/owl/owls/NewClientProcess.owl#ErrorValue=Ok!
Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/AcquireRoleProcess.owl
 RoleID=customer UnitID=TourismMarket AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: SearchService =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/SearchServiceProcess.owl
 SearchServiceInputServicePurpose=SearchCheapHotel"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
Client: OOH! SF Has agreed to excute the service SearchService!
```

```
Client: GetProfile =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/GetProfileProcess.owl
 GetProfileInputServiceID=29"
 :protocol  fipa-request
)
Client: SF SearchService IDProfile = 29
Client: OOH! SF Has agreed to excute the service GerProfile!
Client: SF GetProfile URL = http://localhost:8080/SearchCheapHotel/owl/owls/SearchCheapHotelProfile.owl

Client: AcquierRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/AcquireRoleProcess.owl
 RoleID=customer UnitID=tourismmarket AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: GetProcess =(REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
 GetProcessInputServiceID=29"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service AcquireRol!
Client: OMS AcquiereRol Status = Error ErrorValue = Duplicate
Client: OOH! SF Has agreed to excute the service GetProcess!
Client:SF has informed me of the status of my request. They said :
GetProcessProcess=http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
#GetProcessOutputProcessList=29@28-ManagerAgency1
http://localhost:8080/SearchCheapHotel/owl/owls/SearchCheapHotelProcess.owl,
http://localhost:8080/sfservices/SFservices/owl/owls/GetProcessProcess.owl
#GetProcessOutputServiceStatus=1
Client: SF GetProcess Provider = ManagerAgency1
Client: Ejecutando servicio...
[Client]Sms to send: (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name ManagerAgency1@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/SearchCheapHotel/owl/owls/SearchCheapHotelProcess.owl
 SearchCheapHotelInputCategory=categoria SearchCheapHotelInputCountry=España
 SearchCheapHotelInputCity=Valencia"
 :protocol  fipa-request
)
[Client]Sending...
Client: OOH! ManagerAgency1 Has agreed to excute the service GenericService!
Client:ManagerAgency1 has informed me of the status of my request. They said :
SearchCheapHotelProcess=
http://localhost:8080/SearchCheapHotel/owl/owls/SearchCheapHotelProcess.owl
#SearchCheapHotelOutputHotel=Hotel Puerta de Valencia,
http://localhost:8080/SearchCheapHotel/owl/owls/SearchCheapHotelProcess.owl
#SearchCheapHotelOutputHotelCompany=NH
```

```
[Client] Valores devueltos.
      SearchCheapHotelOutputHotelCompany = NH
      SearchCheapHotelOutputHotel = Hotel Puerta de Valencia
Client: Salir de THOMAS
Client: LeaveRol = (REQUEST
 :sender  ( agent-identifier :name Client@bowser:1099/JADE
 :addresses (sequence http://bowser.dsic.upv.es:7778/acc ))
 :receiver  (set ( agent-identifier :name OMS@localhost
 :addresses (sequence http://localhost:8081 )) )
 :content  "http://localhost:8080/omsservices/OMSservices/owl/owls/LeaveRoleProcess.owl
 RoleID=member UnitID=virtual AgentID=Client@bowser:1099/JADE"
 :protocol  fipa-request
)
Client: OOH! OMS Has agreed to excute the service LeaveRol!
Client: OMS AcquiereRol Status = Ok ErrorValue = <ErrorValue/>
```

Bibliografía

[1] E. Argente, V. Botti, C. Carrascosa, A. Giret, V. Julian, and M. Rebollo. An Abstract Architecture for Virtual Organizations: The THOMAS approach. *Knowledge and Information Systems*, pages 1–35, 2011.

[2] E. Argente, J. Palanca, G. Aranda, V. Julian, V. Botti, A. García-Fornes, and A. Espinosa. Supporting Agent Organizations. In *5th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'07)*, volume 4696 of *LNAI*, pages 236–245. Springer, 2007.

[3] Kaponis D. Artikis A. and Pitt J. *Dynamic Specifications of Norm-Governed Systems*, chapter Multi-Agent Systems: Semantics and Dynamics of Organisational Models. V. Dignum, 2009.

[4] Guido Boella, Joris Hulstijn, and Leendert W. N. Van Der Torre. Virtual organizations as normative multiagent systems. In *Hawaii International Conference on System Sciences*, 2005.

[5] C. Carrascosa, A. Giret, V. Julian, M. Rebollo, E. Argente, and V. Botti. Service Oriented Multi-agent Systems: An open architecture. In *Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1291–1292, 2009.

[6] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multi-agent systems. In *Agent-Oriented Software Engineering*, pages 214–230, 2003.

[7] Ian T Foster. The anatomy of the grid: Enabling scalable virtual organizations. In *Cluster Computing and the Grid*, volume cs.AR/0103, pages 6–7, 2001.

[8] A. Giret, V. Julian, M. Rebollo, E. Argente, C. Carrascosa, and V. Botti. An Open Architecture for Service-Oriented Virtual Organizations. In *PROMAS 2009 Post-Proceedings*, pages 1–15. Springer, 2010.

[9] M. Luck and P. McBurney. Computing as interaction: Agent and agreement technologies. In *IEEE SMC Conf. Distributed Human Machine Systems*, pages 1–6, 2008.

[10] A. Machado and V. Julian. Supporting Dynamics Multiagent Systems on Thomas. In *9th International Conference on Practical Applications of Agents and Multiagent Systems*, volume 89, pages 167–174. Springer-Verlag Berlin Heidelberg, 2011.

[11] Xinjun Mao and Eric Yu. Organizational and social concepts in agent oriented software engineering. In *IN AGENT-ORIENTED SOFTWARE ENGINEERING V. LNCS 3382*, pages 1–15. Springer, 2004.

[12] A. Mas. *Métodos y Herramientas*, chapter 1. Pearson Educación, 2005.

[13] E. Del Val, N. Criado, C. Carrascosa, V. Julian, M. Rebollo, E. Argente, and V. Botti. THOMAS: A Service-Oriented Framework For Virtual Organizations. In *9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1631–1632, 2010.

[14] Estefanía Argente Villaplana. *GORMAS: Guías para el desarrollo de Sistemas Multiagente abiertos basados en organizaciones*. PhD thesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2008.